Vysoká škola Báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Algoritmy pro rozpoznávání barevných kódů rezistorů

Algorithms for Resistor Color Code Recognition

2014 **Bc. Lukáš Rygol**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Lukáš Rygol**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T059 Mobilní technologie

Téma: Algoritmy pro rozpoznávání barevných kódů rezistorů
Algorithms for Resistor Color Code Recognition

Zásady pro vypracování:

Cílem diplomové práce je navrhnout a implementovat algoritmy pro automatické rozpoznávání hodnot rezistorů na základě jejich barevného značení. Vstupem je obraz zachycující odečítaný rezistor v dostatečné blízkosti. Výstupem bude hodnota rezistoru a jeho tolerance. Aplikace na základě analýzy obrazu nalezne a vyhodnotí barevné značení rezistoru. Rezistor může být v prostoru libovolně orientován. Předpokládá se čtyř i pěti proužkové značení.

1. Rešerše algoritmů pro čtení barevných značení (nejen rezistorů).
2. Detekce rezistoru a jeho orientace v prostoru.
3. Nalezení vhodného hranového detektoru.
4. Implementace vlastního řešení s využitím vhodné techniky.
5. Testování algoritmu na různých typech rezistorů v rozdílných podmínkách snímání.
6. Závěrečné kritické shrnutí dosažených výsledků

Seznam doporučené odborné literatury:

James Steele, Nelson To, The Android Developer's Cookbook: Building Applications with the Android SDK, Addison-Wesley Professional, 2010, ISBN-13: 978-0321741233
Rafael C. Gonzalez, Digital Image Processing, Prentice Hall, 2007, ISBN-13: 978-0131687288
Horstmann, C. S, Core Java Volume I--Fundamentals (9th Edition), Prentice Hall; 9 edition, 2012, ISBN 0137081898

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Ing. Michal Krumnikl**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * OSTRAVA * Fakulta elektrotechniky a informatiky

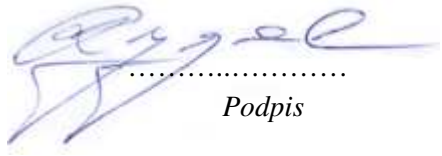
doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: *7. 5. 2014*



……..……

Podpis

Poděkování

Rád bych poděkoval *Mgr. Ing. Michalu Krumniklovi* za odbornou pomoc a konzultaci při vytváření této diplomové práce. Dále bych rád poděkoval rodině a svým nejbližším za psychickou a morální podporu.

Abstrakt

Cílem této diplomové práce je navrhnout algoritmy v programovacím jazyce Java, které by byly schopné co nejvhodněji detekovat rezistor a vyhodnotit jeho ohmickou hodnotu. První část se zabývá značením elektronických součástek. Následně se hovoří o algoritmech pro rozpoznávání barev a současných řešeních aplikací pro čtení barevných kódů rezistorů. Dále se pojednává o operačním systému Android, na kterém se budou dané algoritmy testovat. A na závěr se zde zmiňuje o obrazovém zpracování. Čtení barevného značení a vyčítání hodnoty z tabulek je zdlouhavé a nepohodlné. To vede k automatizovanému řešení, kterých v současné době není příliš mnoho. Ty, které existují, nedosahují příliš dobrých výsledků, a to je podnětem k vytvoření aplikace, která by shrnovala co možná nejlepší řešení. Aplikace, které existují, většinou staticky oříznou obraz nebo musí mít určité natočení elektronické součástky. V prvním kroku je potřeba nalézt co nejvhodnější hranový detektor a z toho můžeme získat oblast zájmu a natočení součástky. Dále je potřeba využít vhodného algoritmu pro výčet barevných proužků a jejich správnou valuaci.

Klíčová slova

Android; Java; algoritmy; rezistor; obrazové zpracování; hranový detektor

Abstract

The goal of this diploma thesis is to implement an algorithm that would be able to recognize the resistor markings and evaluate its value. The work will be implemented in Java for Android OS. The first part describes the various types of resistors and electronic components markings. Furthermore we will discuss the algorithms which can recognize color codes and current solutions which are able to recognize the resistor markings. Furthermore we will describe Android OS. In the end we will discuss the image processing. Reading the color codes on the components and evaluate their values from reference color tables takes a lot of time and it is uncomfortable. These problems can be solved by an automated system. There are only a few available solutions, and they do not provide good results. This is the reason for creating our own application which will provide better solutions. The existing applications usually cut the static region of interest, in which resistor must be placed. In the first step I have to find the best edge detector. After that I have to find the region of interest and detect the rotation angle. We will also need to find a suitable algorithm which will enumerate color code and calculate the correct values.

Key words

Android; Java; resistor; image processing; edge detector

Seznam použitých zkratek

Zkratka	Anglický význam	Český význam
1D	One-dimensional	Jednorozměrný
2D	Two-dimensional	Dvourozměrný
ADT	Android Development Tool	Vývojový nástroj pro Android
API	Application Programming Interface	Rozhraní pro programování aplikací
BSD	Barkeley Software Distribution	Barkeleyho softwarová distribuce
CDT	C/C++ Development Tools	C/C++ vývojové nástroje
CIE	The International Commission on Illumination	Mezinárodní komise pro osvětlování
CMYK	Cyan Magneta Yellow Key	Azurová, purpurová, žlutá, černá
DIN	Deutsches Institut für Normung	Německý ústav pro průmyslovou normalizaci
EIA	Electronic Institut Alliance	Aliance elektronického průmyslu
GPS	Global Positioning System	Globální poziční systém
HSV	Hue, Saturation, Value	Barevný tón, sytost barvy, hodnota jasu
JDT	Java Development Tools	Vývojové nástroje pro Javu
JIT	Just-in-time	Kompilátor pro zrychlení běhu
JNI	Java Native Interface	Přirozené rozhraní Javy
LED	Light Emitting Diode	Typ světelného zdroje
MELF	Metal Electrode Leadless Face	SMD s pouzdrem válcového tvaru
NTSC	National television System Committee	Televizní standard
OS	Operating System	Operační systém
PAL	Phase Alternation Line	Televizní standard
PDT	PHP Developer Tools	PHP vývojové nástroje
RGB	Red Green Blue	Červená, zelená, modrá
ROI	Region of interest	Oblast zájmu

SMD	Surface Mount Device	Součástky pro povrchovou montáž
TTS	Text-To-Speech	Nástroj pro syntézu řeči
USB	Universal Serial Bus	Univerzální sériová sběrnice
VM	Virtual Machine	Virtuální stroj
VPN	Virtual Private Network	Virtuální privátní síť
XML	eXtensible Markup Language	Rozšířitelný značkovací jazyk
YCbCr	Brightness, Blue Chroma, Red Chroma	Jas, modrá chrominance, červená chrominance
YIQ	Birghtness, In-Phase, Quadrature	Jas, činná složka, kvadrát
YUV	Brightness, Two chrominance components	Jas, dvě barevné složky

Obsah

Úvod

V současné době se lidé snaží dělat věci s pomocí nejmodernější techniky a opakované procesy provádět automatizovaně. Proto se pro přístroje vytváří aplikace, které se snaží tuto automatizovanou práci obsluhovat. Jedná se tak z důvodů, kde cílem je redukovat náklady, protože lidská práce je cennější nebo se snažíme ulehčit si práci.

Úkolem této diplomové práce je vytvořit aplikaci, která by automatizovaně četla barevné značení rezistorů v jakémkoli natočení. Počítá se s čtyř-pruhovými i pěti-pruhovými součástkami. K tomu se musí zvolit správné prvky z obrazového zpracování a použít je vhodně v algoritmech pro správné vypočítání konečné hodnoty rezistoru.

Jak již bylo výše zmíněno, tak hlavním principem celé aplikace je obrazové zpracování. Vstupem se myslí sekvence snímků, které zachycují rezistor v dostatečné vzdálenosti. Uvažuje se, že aplikace by mohla fungovat real-time a hodnoty získat bez ukládání fotografií na uložiště v mobilním zařízení. V oblasti digitálního zpracování obrazu se objevuje mnoho aspektů, které mohou mít dopad na konečný výsledek. Jedná se především o změnu intenzity osvětlení nebo pozadí, na kterém se objekt vyskytuje.

Na začátku práce se seznámíme se značením pasivních elektronických součástek, konkrétně s rezistory a kondenzátory. Pak si popíšeme některé metody a funkce pro barevné rozpoznávání. Následně rozeberme některá současná řešení, která se snaží pomocí moderních zařízení vypočítat z barevného značení rezistoru jeho ohmickou hodnotu. Android je platforma pro mobilní zařízení a my si v této práci představíme verze, které se ještě v současné době používají, jejich rozdíly a důvod, proč byl zvolen tento OS. Další kapitola se věnuje obrazovému zpracování. Najdeme zde postupy a principy fungování nejrůznějších metod.

Praktická část ukazuje postupy, kdy se aplikují jednotlivé části obrazového zpracování pro získání ohmické hodnoty součástky. Je zde vysvětleno, proč a jaký se použil hranový detektor, proč se použila Houghova transformace pro vyhledávání přímek v obraze, jak se získal obrys rezistoru, až po získání hodnoty z výpočtu barevných pruhů.

Na závěr můžeme najít výsledné testy, kde je zobrazena úspěšnost rozpoznávání jednotlivých barev při různých intenzitách světla. Zároveň se zde nachází kritické shrnutí dosažených výsledků.

1 Značení součástek

Elektronické součástky se značí podle jejich druhu nebo velikosti a to většinou na základě příslušné mezinárodní normy, která definuje přesný tvar jak má značení vypadat [5]. Některé prvky se značí barevným značením a jiné zase písmenky a číslicemi. Barevné značení rezistorů, kondenzátorů a v některých případech cívek se používá norma EIA-RS-279.

1.1 Značení rezistorů

Rezistory, které se vyrábějí, se označují v souladu s mezinárodně normovanými řadami začínající písmenem E a dále číslicemi, jež symbolizují počet členů v každé dekádě. Máme-li například E6, tak nám název udává, že obsahuje v každé dekádě následujících šest hodnot nebo jejich dekadické násobky: 1 - 1,5 - 2,2 - 3,3 - 4,7 - 6,8 (řady jmenovitých hodnot můžeme nalézt v katalogu). V současnosti se využívají řady E3, E6, E12, E24, E48, E96 a E192. Pokud nám chybí odpor nějaké hodnoty, tak se pokusíme sériovým, paralelním nebo kombinovaným zapojením sestavit příslušnou ohmickou velikost. Rezistory se značí třemi způsoby. Tabulky 1.1 a 1.2 zobrazují jmenovité hodnoty, které patří do dané řady. Tyto hodnoty [20] se dále násobí mocninou desítky.

Tabulka 1.1: Hodnoty obsažené v řadě E12

1,0	1,2	1,5	1,8	2,2	2,7	3,3	3,9	4,7	5,6	6,8	8,2

Tabulka 1.2: Hodnoty obsažené v řadě E24

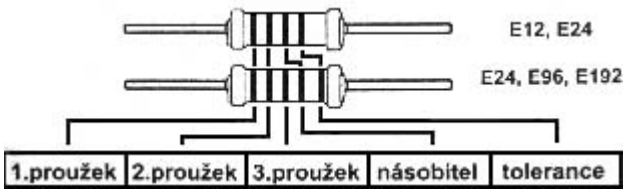
1,0	1,1	1,2	1,3	1,5	1,6	1,8	2,0	2,2	2,4	2,7	3,0
3,3	3,6	3,9	4,3	4,7	5,1	5,6	6,2	6,8	7,5	8,2	9,1

1.1.1 **Klasické součástky**

Značí se číslicemi a písmeny. TR 191 4k7/A, kde TR znamená udávanou součástku v tomto případě typizovaný rezistor. Číslicí 191 je udáváno provedení součástky a další charakteristiky můžeme nalézt v katalogu součástek. 4k7 je jmenovitá hodnota odporu a v tomto případě znamená 4700 Ω. Písmeno na konci značí toleranci.

1.1.2 **Rozměrově malé součástky**

V některých případech není příliš vhodné použít výše uvedené kódování. Takový případ nastane například, když chceme číst údaje z různých směrů, potom tedy použijeme barevné značení jmenovitých hodnot. Značení se provádí barevnými pruhy po obvodu součástky (Obr. 1.1). Kódy můžou být čtyř a pěti pruhové [5]. Ty se čtou od takového pruhu, který se nachází co nejblíže k vývodu rezistoru. U čtyř-proužkového první a druhá hodnota znamená hodnotu, třetí násobitel a poslední toleranci. U pěti-proužkového první, druhá a třetí značí číslici, jinak zbytek zůstává stejný. Tabulka 1.3 zobrazuje význam jednotlivých barev.



Obr. 1.1: Význam jednotlivých proužků u rezistoru

http://www.tydyt.cz/kabely/rozhrani/rezistor_1.jpg

Tabulka 1.3: Význam jednotlivých barev ve značení

Barva	1. proužek	2. proužek	3. proužek	Násobitel	Tolerance
Černá	0	0	0	10^0	
Hnědá	1	1	1	10^1	F (± 1 %)
Červená	2	2	2	10^2	G (± 2 %)
Oranžová	3	3	3	10^3	
Žlutá	4	4	4	10^4	D (± 0,5 %)
Zelená	5	5	5	10^5	C (± 0,25 %)
Modrá	6	6	6	10^6	B (± 0,1 %)
Fialová	7	7	7	10^7	
Šedá	8	8	8	10^8	
Bílá	9	9	9	10^9	
Zlatá				10^{-1}	J (± 5 %)
Stříbrná				10^{-2}	K (± 10 %)
Žádná					M (± 20 %)

1.1.3 SMD součástky

Pokud se jedná o čipové provedení součástky, tak čteme hodnotu z číslic. Pokud se jedná o válcové rezistory typu MELF, tak ty se značí barevnými čárkovými kódy. U prvně zmiňovaného případu jsou rozměry miniaturní, a proto na povrchu můžeme nalézt pouze dvě nebo tři číslice udávající hodnotu a poslední znamená násobitele. Pokud chceme uvést desetinou čárku, tak se ve značení musí vyskytnout R.

1.2 Značení kondenzátorů

Kondenzátory různých kapacit jsou dodávány v hodnotách podle předem určených řad. Bývají vyráběny různými výrobci a ti využívají odlišná značení (Obr. 1.2). Proto musíme být při výběru opatrní. Značení se skládá z několika čísel, která udávají jmenovitou hodnotu, provozní napětí a toleranci [5].

1.2.1 Značení kondenzátorů dle DIN

Pokud uvidíme na kondenzátoru značení například MKS, tak se jedná o polystyrénový kondenzátor. Další významy jednotlivých písmen nalezneme níže.

M – kovová vrstva

K – plast

Třetí písmeno definuje dielektrikum

S – polystyren

P – polypropylen

C – polykarbonát

T – polyester

U – celuózoacetát

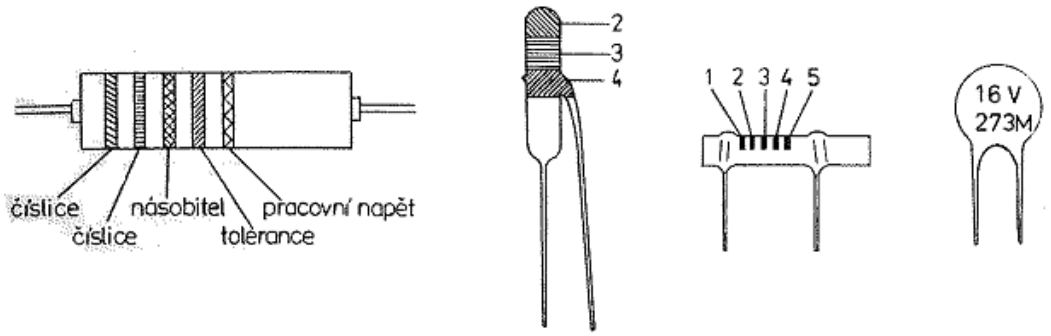
Pokud najdeme na součástce číselné označení, tak symbol, který označuje velikost kapacity, se vytváří podobně jako u rezistorů. Základní jednotkou je zde pikofarad značený písmenem J, k značí tisíc pikofaradů, M - mikrofarad a G - tisíc mikrofaradů. Tolerance následuje za kódem kapacity a tvoří se velkými písmeny. Maximální provozní napětí se udává ve voltech.

1.2.2 **Barevný kód**

Barevné značení na kondenzátorech se uskutečňuje pomocí proužků nebo teček. Toto značení se ve velké míře podobá kódu rozměrově malých rezistorů. U kondenzátorů záleží na provedení, protože podle toho se volí i umístění značek. Tabulka 1.4 vysvětluje významy jednotlivých proužků.

Tabulka 1.4: Význam jednotlivých pruhů u kondenzátorů

Barva	1. kroužek 1. číslice	2. kroužek 2. číslice	3. kroužek 3. číslice	4. kroužek tolerance	5. kroužek pracovní nap.
Černý	0	0	10^0		-
Hnědý	1	1	10^1	1 %	100 V
Červený	2	2	10^2	2 %	200 V
Oranžový	3	3	10^3		300 V
Žlutý	4	4	10^4		400 V
Zelený	5	5	10^5		500 V
Modrý	6	6	10^6		600 V
Fialový	7	7	10^7		700 V
Šedý	8	8	10^8		800 V
Bílý	9	9	10^9		900 V
Zlatý				5 %	1000 V
Stříbrný				10 %	2000 V
Bezbarvý				20 %	-



Obr. 1.2: Značení hodnot na kondenzátorech

1.2.3 **Číselné značení**

Tyto kódy se používají například u keramických kondenzátorů a význam jednotlivých hodnot najdeme v tabulce 1.5.

Tabulka 1.5: Význam značení u kondenzátorů s číslicemi

Kapacita		Násobitel	Tolerance	
1. číslice	2. číslice	3. číslice	Písmeno	
1	1	$1 = 10^1$	C	± 0,25 pF
2	2	$2 = 10^2$	D	± 0,5 pF
3	3	$3 = 10^3$	F	± 1 pF
4	4	$4 = 10^4$	G	± 2 %
5	5	$5 = 10^5$	J	± 5 %
6	6	-	K	± 10 %
7	7	-	M	± 20 %
8	8	-	P	- 0 / + 100 %
9	9	-	S	- 20 / 50 %
0	0	-	-	-

1.2.4 **SMD**

U tohoto typu kód nebývá, protože se počítá s tím, že kondenzátory SMD jsou určeny pro automatické osazování plošných spojů přímo ve výrobě. Je-li keramický kondenzátor označen, tak se na něm vyskytuje jedna číslice a písmeno nebo značení se skládá ze tří číslic [5].

2 Algoritmy pro rozpoznání barevného značení

Všeobecně algoritmy, které popisují rozpoznání dvou barev, předpokládají, že existuje někde hodnota referenční nebo se klasifikuje podle určitého prahu, kdy určíme, že daná barva spadá do určité množiny. Dále záleží na použité technice pro získání obrazu a barevném prostoru, protože v některých situacích může být vhodnější používat jiný model než původní. Nyní si popíšeme několik algoritmů, které provádí vyhodnocování barev.

2.1 Metoda vzdálenosti vektoru

Tato metoda pracuje na základě vzdálenosti dvou vektorů a vypočítá se vzorcem (2.1). Představme si, že hledáme zelenou barvu, tak potom $R_2 = 0$; $G_2 = 255$; $B_2 = 0$. Hodnoty jsou uvedeny jako ideální, a proto počítáme s tím, že do tohoto výpočtu musíme zahrnout i některé další odstíny zelené. Hodnoty R_1, G_1, B_1 se získají z daného pixelu.

$$D = \sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2} \tag{2.1}$$

Na základě uvedeného vzorce (2.1) můžeme naprogramovat funkci, která nám takový rozdíl barev vrací. Uvažujeme barevný model RGB, kde jednotlivé složky reprezentují hodnoty získané v daném pixelu.

Nyní si zvolíme práh P, který nám zaručí, že přijmeme i jiné odstíny zelené. Procházením u všech pixelů v obraze kontrolujeme, že vrácená hodnota D je menší než práh P.

2.2 Detekce pomocí jiných prostorů

V oblasti počítačového vidění se běžně pracuje s barevným prostorem RGB. Zařízení používaná ve videotechnice používají barevný model YUV. Lidskému vnímání nejvíce koresponduje barevný prostor HSV. V případě konverze je nutné využít určité vzorce (2.2), zde se demonstruje převod RGB na HSV [27].

$$R' = \frac{R}{255}; G' = \frac{G}{255}; B' = \frac{B}{255}$$

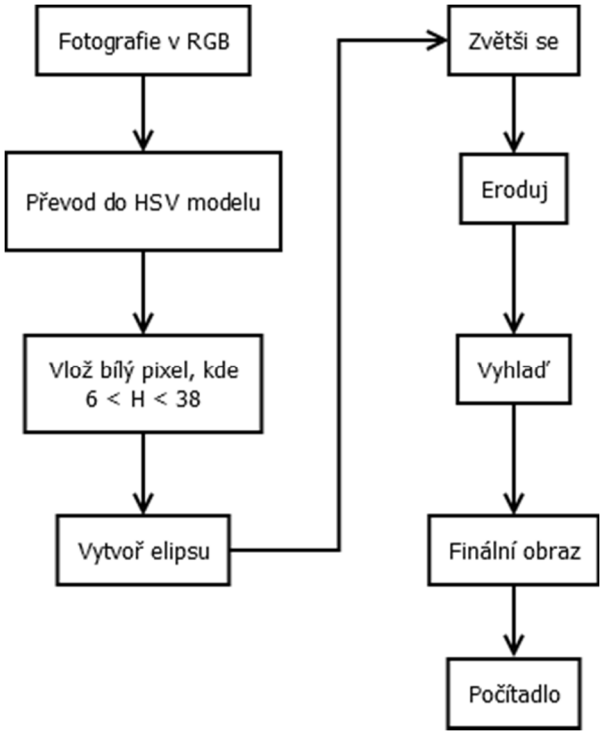
$$\Delta = C_{max} - C_{min}; C_{max} = \max(R', G', B'); C_{min} = \min(R', G', B')$$

$$H = \begin{cases} 60° \times \left(\frac{G'-B'}{\Delta} mod 6\right), C_{max} = R' \\ 60° \times \left(\frac{B'-R'}{\Delta} + 2\right), \quad C_{max} = G' \\ 60° \times \left(\frac{R'-G'}{\Delta} + 4\right), \quad C_{max} = B' \end{cases} \tag{2.2}$$

$$S = \begin{cases} 0, & \Delta = 0 \\ \frac{\Delta}{C_{max}}, & \Delta <> 0 \end{cases}$$

$$V = C_{max}$$

Článek [23] pojednává o vyhledávání lidské kůže na digitální fotografii. K tomuto účelu se používá HSV barevný model. Využívá H kanál pro zjištění, jestli daný pixel patří do přijímací množiny či nikoli.



Obr. 2.1: Kroky algoritmu

Převzato z: [23]

Hodnota 3 – 36 pro H se zjistila testy, kdy se vkládaly různé fotografie různých lidí.

Vzdálenost mezi dvěma barvami se dříve popisovala pouze adjektivními barvami. Více o této problematice se můžeme dočíst v [28]. Nyní lze využít vzdálenost v Euklidovském prostoru nezávisle na barevném prostoru.

2.2.1 **Delta E**

Mezinárodní komise pro osvětlování, též známá pod zkratkou CIE, představila svou metrickou vzdálenost pod názvem Delta E. Vzorce CIE76 [28] pracují s hodnotami Lab. Další úpravy vzorce proběhly v roce 1994 a 2000. Ukázalo se totiž, že předešlý vzorec není tak vjemově jednotný a to zejména v oblastech nasycených. To znamená, že vzorec obsahoval míry barev příliš vysoko. Vzorec je vyobrazen (2.3), dále můžeme vidět následné modifikace ve vzorcích (2.4) a (2.5).

$$\Delta E^*_{ab} = \sqrt{(L^*_1 - L^*_2)^2 + (a^*_1 - a^*_2)^2 + (b^*_1 - b^*_2)^2}, \tag{2.3}$$

kde L značí světelnost, a je osa zelná – červená, b je osa modrá – žlutá.

$$\Delta E^*_{94} = \sqrt{\left(\frac{\Delta L^*}{k_L S_L}\right)^2 + \left(\frac{\Delta C^*_{ab}}{k_C S_C}\right)^2 + \left(\frac{\Delta H^*_{ab}}{k_H S_H}\right)^2} \tag{2.4}$$

Význam jednotlivých prvků můžeme nalézt v [12]. Jak již bylo psáno výše, tak poslední úprava vztahu se konala v roce 2000 (2.5). Bylo to z důvodu, že verze z roku 1994 (2.4) neřešila dostatečně vnímání problému pro jednotnost, a proto nový vzorec obsahoval pět oprav.

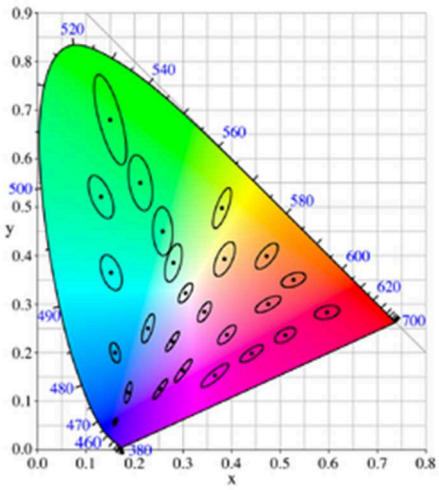
$$\Delta E^*_{00} = \sqrt{\left(\frac{\Delta L'}{k_L S_L}\right)^2 + \left(\frac{\Delta C'}{k_C S_C}\right)^2 + \left(\frac{\Delta H'}{k_H S_H}\right)^2 + R_T \frac{\Delta C'}{k_C S_C} \cdot \frac{\Delta H'}{k_H S_H}} \tag{2.5}$$

Vysvětlení jednotlivých parametrů nalezneme v [13]. Ty se dále rozkládají do dílčích výpočtů. Poslední modifikace vzorce řeší pět následujících problémů:

- Rotace odstínu R_T, tím se vypořádáme s problematickými oblasti modré (konkrétně u odstínů, kde je úhel 275°)
- Kompenzace neutrálních barev (převedení hodnot do rozdílů L*C*h)
- Náhrada za světelnost (S_L)
- Kompenzace za barevnost (S_C)
- Náhrada za odstín (S_H)

2.3 Tolerance

Tolerance řeší otázku, které barvy spadají ještě do množiny okolo referenční barvy [1]. Pokud tedy vzdálenost mezi dvěma barvami nepřesáhne práh, tak lze označit tento bod do přijímací množiny. Například v CIE 1931 barevný prostor obsahuje toleranční obrysy, které jsou definovány dle MacAdama. Jak lze pozorovat na (Obr. 2.2), tak elipsy nemají stejnou velikost, a to vede k vytvoření CIELUV a CIELAB.



Obr. 2.2: MacAdamův diagram

Převzato z: www.justinmonahan.com/architecture/wiki/images/f/f4/CIExy1931_MacAdam.png

2.4 Shrnutí

U většiny algoritmů, které byly zveřejněny, se počítá s tím, že se v obraze hledá barva, která odpovídá co nejblíže referenční. V reálném životě může barva, kterou hledáme, mít jiný odstín. To závisí na intenzitě osvětlení a dalších aspektech. V takovém případě uvažujeme toleranci, abychom barvu správně ohodnotili. Někdy je lepší pracovat nad jiným barevným prostorem, které jsou pro lidské vnímání příznivější, postup detekce je ale obdobný. Některé barvy však dělají problém i lidskému oku, proto je nutné počítat i v obrazovém zpracování s odchylkami a špatným ohodnocením. Nápad, jak omezit chybovost je, že použijeme algoritmus, který se snaží porovnávat výsledné hodnoty s tabulkou, která bude obsahovat řady rezistorů - E12, E24, atd. Kdyby se v takovém případě rozhodovalo mezi hodnotou 440 ohmů a 470 ohmů, tak pravděpodobnější hodnota bude 470 ohmů, protože patří do řady.

3 Současná řešení

V současné době existuje několik řešení, která se snaží vypočítat hodnotu rezistoru na základě obrazového zpracování. Z důvodu nepříjemného a zdlouhavého vyčítání hodnot z barevných tabulek by znamenalo zjednodušení, kdyby se vytvořila aplikace, která by detekovala barevné značení automatizovaně například pomocí mobilního telefonu. Níže si popíšeme, jak byly některé aplikace řešeny, kde tkví případné nedostatky a jak by se daly eliminovat. Některé případy jsou ne příliš pohodlné, protože nutí uživatele zamýšlet se, jestli je například rezistor správně umístěn a jsou-li splněny všechny podmínky pro správné přečtení kódu.

3.1 Metody pro čtení barevného značení

Postupy a metody pro vyhodnocení barevných pruhů u rezistorů můžeme nalézt v [18]. Zde můžeme naleznout článek, který se přímo zabývá tím, jak by se mělo postupovat k získání obrazové informace vedoucí k výsledné hodnotě elektronické součástky. Avšak není zde uvedena konkrétní implementace v nějakém specifickém programovacím jazyce, čili tento postup může pouze inspirovat tvůrce.

Uvádí se zde, že pro získání barevných polí, se používá obrazové segmentace. Pro získání hodnot v daném regionu se aplikuje metoda k-průměrů. Toto vše probíhá na základě tří vektorů. Dále se vytvoří 1-NN klasifikátor, který klasifikuje barvu určené plochy.

Experimenty se prováděly za tří světelných podmínek a to pří nízké intenzitě osvětlení (100 lx), průměrném osvětlení (3000 lx) a velmi vysokém osvětlení (6000 lx). Barevný prostor se nepoužíval jen RGB, ale i L*a*b* a L*u*v*.

Ukázka v článku experimentovala s čtyř-proužkovou součástkou a pracovalo se s 11 barvami, jež se používají ke značení. Nepočítá se se stříbrnou barvou a situací, kdy je pruh vynechán, protože takové případy jsou ojedinělé. Pozice rezistoru byla uprostřed a jeho hodnota se určovala z levé strany. Od každé světelné simulace bylo pořízeno třicet snímků, se kterými se experimentovalo. Rozlišení takových obrazů se pohybovalo okolo 500 x 250 pixelů.

Pro získání barevných pruhů se používají následující techniky. V první řadě se pořídí snímek rezistoru, u kterého chceme zjistit jeho ohmickou hodnotu. Potom získáme pozadí daného obrazu (vypadá to, že pozadí by mělo být idealizované a jednobarevné pro snazší rozpoznání). To se provede tak, že se vezmou pixely, které se nachází na souřadnicích (10, 10; šířka obrazu - 10, 10; 10, výška obrazu - 10 a poslední šířka obrazu - 10 a výška obrazu - 10). Těmito čtyřmi body se získalo pozadí. Následně se fotografie převedla do monochromatického obrazu. Z toho se odvodily horizontální a vertikální histogramy. Prahováním těchto histogramů metodou založené na analýze diskriminantu, získáme tělo rezistoru. Pro další práci se vystřihl zájmový region o výšce 15 pixelů (šířka zůstala stejná). Avšak v tomto se vyskytl problém, a to takový, že každý pixel měl rozdílnou barvu. To se řešilo pomocí seskupování podobných barev. Pro výpočet konečné hodnoty pruhu se použije metoda k-průměru vektoru w, vektoru pozice u a vektoru barvy v. Výpočet jednotlivých vektorů můžeme vidět na vzorci (3.1).

$$w = (tx.(1-t)r, (1-t)g.(1-t)b)^T, 0 \leq t \leq 1$$
$$u = (x)^T \quad (3.1)$$
$$v = (r, g, b)^T$$

Kde *t* značí váhu kombinačního vektoru, *x* je pozice pixelu z levé strany. Kombinační vektor je v podstatě ekvivalentní pozičnímu vektoru, když *t* = 1 a když u barevného vektoru *t* = 0. RGB prostor se rozdělí do *n* x *n* x *n* podprostorů. V každém podprostoru je počítán vzorek. Vybralo se *k* podprostorů v sestupném pořadí a vypočítal se jejich průměrný vektor. V tomto experimentu bylo dosazeno *n* = 12, *k* = 12.

Ze získaných barevných oblastí se musí klasifikovat správná hodnota. Pro tento účel se vytvoří 1-NN klasifikátor, který je také známý jako pole pro rozpoznávání vzorků. Výsledky jsou zobrazeny v tabulce 3.1.

Tabulka 3.1: Výsledky správné klasifikace barev

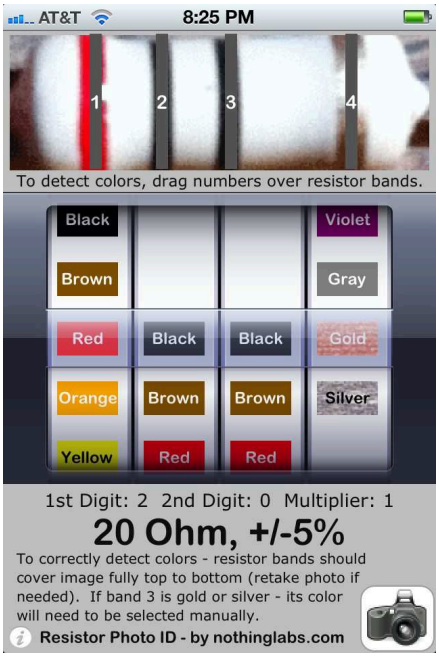
Světelnost	RGB model	L*a*b* model	L*u*v* model
100 lx	2 / 17	6 / 17	3 / 17
3000 lx	17 / 24	16/ 24	21 / 24
6000 lx	22 / 28	24 / 28	23 / 28

3.2 Resistor Photo ID

První z takových aplikací [26] je program navržený pro mobilní platformu iOS, bohužel s vývojem nových verzí tohoto operačního systému již není možné danou aplikaci používat z důvodů nekompatibility.

Program využívá integrovaného fotoaparátu mobilního zařízení. Při pořízení snímku je nutné dodržet některá pravidla pro správné fungování. První z nich je, že se můžou použít pouze ty rezistory, které obsahují jen čtyři pruhy. Další podmínkou pro správné rozpoznání je umístění elektronické součástky do předem vyznačeného pole. Toleranční proužek se musí nacházet na pravé straně. Intenzita osvětlení by měla být vhodně zvolená. Pokud se hodnota pruhu špatně detekuje, tak se dá nahradit v posuvnících.

Program se vyvíjí pod firmou Nothing Labs, avšak zdrojové kódy nebo postup řešení aplikace nebyl nikde zveřejněn. V současné době se již dále nevyvíjí a na původním oficiálním úložišti tzv. iTunes ji není možné nadále stáhnout. Z tohoto důvodu se aplikace jako taková testovat nemohla, a tudíž není možné ani posoudit úspěšnost rozpoznávání. Na (Obr. 3.1) vidíme, jak aplikace vypadala.



Obr. 3.1: Screenshot Resistor Photo ID

http://www.nothinglabs.com/resistorphotoid/shot2.jpg

3.3 ResCan

Další zajímavou aplikací je ResCan. K této práci se dají stáhnout zdrojové kódy, takže tato aplikace se dala analyzovat a pozorovat. Její propracování se nejvíce blíží automatizovanému rozpoznávání barevného značení rezistorů. Autor začal psát aplikaci pro mobilní platformu Android, avšak narazil zde na problém, když fotoaparát je od objektu v malé vzdálenosti. Proto se rozhodl využít stolní počítač pro dosažení lepších výsledků.

Aplikace [25] funguje na obrazovém zpracovávání, kde vstupem je sekvence snímků a ty se následně zpracovávají real-time. Využívá se zde svobodné a otevřené multiplatformní knihovny pro manipulaci s obrazem nazývanou OpenCV. Jelikož se program implementoval na stolní počítač, tak bylo nutné použít webkameru. Pro zajištění konstantního osvětlení webkamera obsahovala 6 LED diod, které disponovaly světlem podobnému dennímu. V první řadě se snaží oddělit rezistor od pozadí a dále detekuje jednotlivé barevné pruhy. Při zobrazování výsledku výpočtu autor používá syntézu řeči, konkrétně FreeTTS a zároveň zobrazuje výsledek přímo do obrazu. Červená čára znázorňuje, kde si algoritmus myslí, že se nachází střed rezistoru.

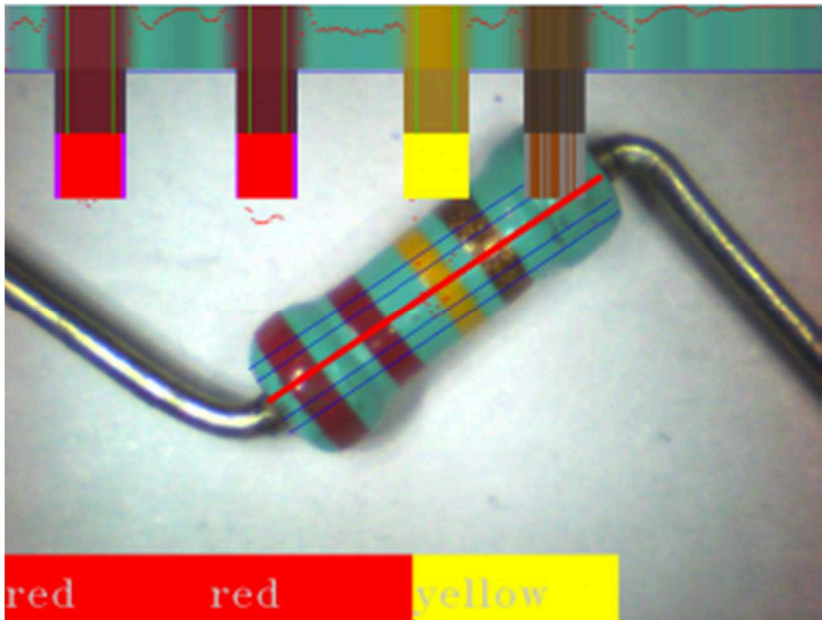
Konkrétně ve zdrojových kódech je postup následující. Hlavní úloha se rozkládá do dvou hlavních metod.

První z nich na začátku převede matici do RGB, vyřeže matici a použije metodu mean, která je definovaná v OpenCV a slouží pro zjištění střední hodnoty v daném poli. Následně prochází obraz a zkouší, jestli se splňuje podmínka, která říká, aby si kód vzal střední hodnotu z pole a zkontroloval, jestli je menší než daný pixel a počítá, kolik takových vzorků odpovídá. Tím si vypočítává oblast zájmu tzv. místo, kde se nachází tělo rezistoru, protože bere jen ty hodnoty, které nepatří do množiny

pro vyhodnocení barev a snižuje tím šířku obrazu. Postup se opakuje i pro výšku obrazu a získá hodnoty min_{x1}, max_{x2}, min_{y1} a max_{y2}. Na konec se zavolá funkce pro detekci.

Druhá funkce vytváří pole o velikosti 3, kde se ukládají hodnoty RGB. Potom prochází obrázek a vykreslí červenou čáru. Z té by se dalo určit úhel naklonění rezistoru. Následně získá oblast zájmu, kde se nachází dané proužky a určí hodnotu pro každý z nich. Na závěr vypíše hodnoty do obrazu a pomocí FreeTTS oznámí výsledek.

Tyto podmínky autor zkoušel na bílém pozadí a tvrdí, že vyhodnocování se pohybuje na dobré úrovni. Při mém testování aplikace na jiném než bílém pozadí dosahovala aplikace vyšší chybovosti. Grafické zpracování je vyobrazeno na (Obr. 3.2).



Obr. 3.2: Ukázka vyhodnocení rezistoru ResCanem

http://armageddon421.de/wp-content/uploads/2014/02/ResCap1.png

4 Android

Android je operační systém založený na upraveném Linuxovém jádře [6] a primárně se instaluje do mobilních zařízení, jako jsou mobilní telefony, tablety a notebooky. V poslední době se ale s ním můžeme setkat i v Set-top boxech a televizorech. Tento OS je distribuován jako open source a obsahuje v sobě operační systém, middleware, aplikace a různá rozhraní. Jádro je napsané v programovacím jazyce C++ a aplikace jsou psané v programovacích jazycích Java nebo C++. Důvodem, proč se tato práce implementuje pro tento operační systém je ten, že je rozšířen a používá se v mobilních zařízeních. Účelem bylo si vybrat platformu, která je moderní a rozšířená na mobilních zařízeních a vývoj pro ni je přívětivý z hlediska publikace aplikace, vývojového prostředí či programovacího jazyka.

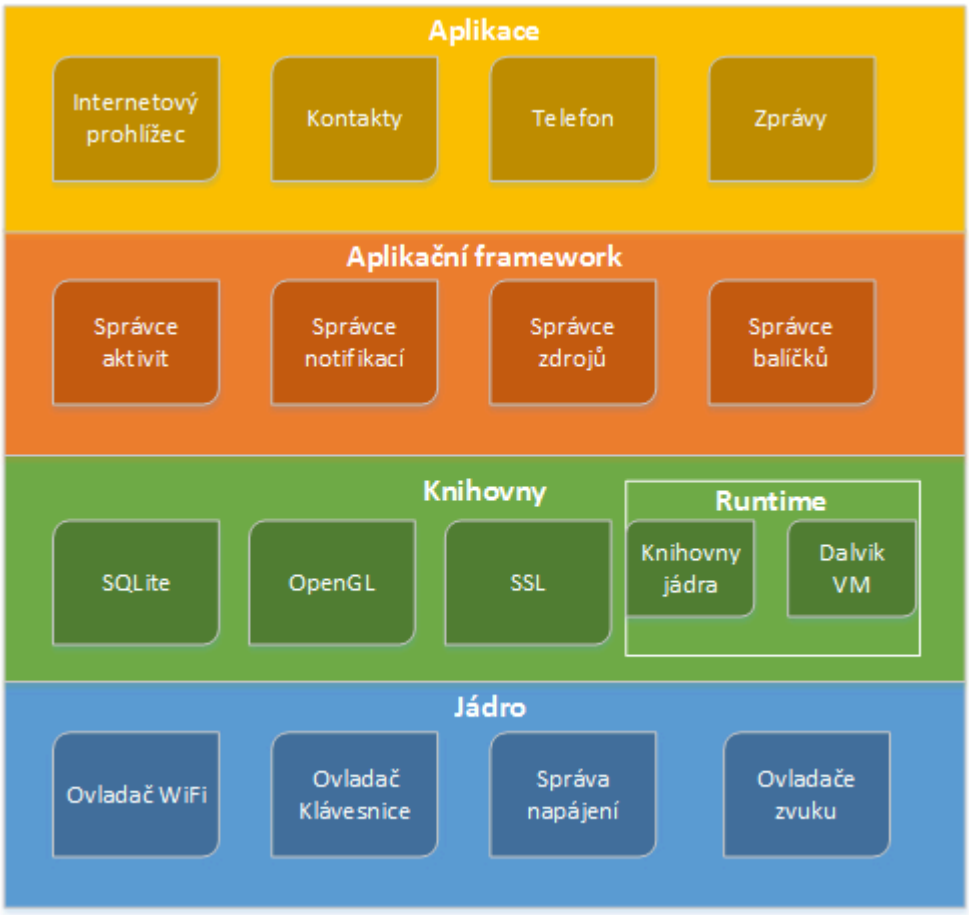
4.1 Historie

Společnost Google Inc. koupila v roce 2005, tehdy ještě neznámou společnost, Android Inc., která byla založena v říjnu roku 2003 autory Andy Rubinem, Richem Minerem, Chrisem Whitem a Nickem Searsem [15]. V roce 2007 společnost získala několik patentů v oblasti mobilních technologií, a tím si připravila půdu pro vstup na trh s mobilními telefony. První uvedení na trh ve verzi Android 1.0 proběhlo 23. září 2008 a prvním zařízením, na kterém daná platforma běžela, bylo HTC Dream [15]. První verze SDK se uvolnila roku 2009. Nyní ho vyvíjí seskupení tvořené mobilními operátory, technologickými firmami, softwarovými společnostmi a výrobci mobilních telefonů. Toto uskupení se nazývá Open Handset Alliance.

4.2 Architektura

Architektura OS se dělí do pěti vrstev (Obr. 4.1). Na nejnižší vrstvě můžeme najít jádro (Linux Kernel) a low level nástroje. O úroveň výše se nachází Knihovny a vrstva Runtime. Následuje Aplikační framework a vrstva, co se vyskytuje nejvýše, se jmenuje Aplikace [6].

Jádro přímo interaguje s hardwarem zařízení [15]. Stejně jako desktopové počítače běžící na Linuxu se jádro zabývá například správou napájení či pamětí, ovladači přístroje, sítěmi a zabezpečením. Knihovny jsou komponenty, které sdílí místo s Runtime komponentou. Slouží jako překladač mezi Jádrem a Aplikačním frameworkem. Jednotlivé knihovny jsou psané v C/C++, ale jsou poskytovány prostřednictvím Java API. To znamená, že vývojáři mohou přistupovat k těmto knihovnám použitím Java aplikačního frameworku. Dalvik Virtual Machine byla napsána Danem Bronsteinem. Účelem této VM je spuštění aplikace na zařízeních s omezenými prostředky. Typicky mobilní telefony, které spadají do této kategorie, obsahují omezení jako menší paměť, kapacita baterie, velikost displeje, atd. Dalvik VM spouští .dex soubory a ty se vytvářejí překladem .class nebo .jar souborů. Dalvik VM patří do Runtime. Aplikační Framework - vrstva, jež je důležitá z hlediska konečného systému nebo end-user aplikace. Poskytuje soubor služeb, které developer potřebuje pro psaní aplikací. Odkazuje na API, které umožňují vytvářet uživatelské rozhraní k aplikacím (button, text box, atd.). Aplikační vrstva - jedná se už o klasické aplikace, které jsou přímo zobrazované koncovému uživateli např. Prohlížeč, Kontakty či Zprávy.



Obr. 4.1: Ukázka Android architektury

4.3 Verze

Detailní popis jednotlivých verzí je možné najít v [3]. Níže jsou uvedeny verze, které se stále používají.

Froyo 2.2.x - Oficiálně představen 20. května 2010 a pracoval s jádrem 2.6.32. Vyznačoval se hlavně optimalizací paměti a JIT kompilací. Podporoval Android Cloud to Device Messaging a umožňoval push notifikace. Z dalších významných novinek byl Wi-Fi hotspot a USB tethering.

Gingerbread 2.3.x - Uvedl se 9. prosince 2010 s jádrem 2.6.35. Obsahuje aktualizaci grafického rozhraní, rozšířenou funkcionalitu kopírování a vkládání, podporuje Near Field Communication, nový download manager a mnoho dalšího.

Ice Cream Sandwich 4.0.x - Funguje na Linux kernelu 3.0.1 a byl publikován 19. října 2011. Má možnost přistupovat k aplikacím ze zamykací obrazovky, disponuje VPN Frameworkem, zabudovaným editorem na úpravu fotek, atd.

Jelly Bean 4.1 - 4.3 - První zařízení nesoucí Jelly Bean byl tablet Nexus 7. Verze 4.1 obsahovala Project Butter, který zvyšoval výkon pomocí předvídání doteku na displeji, rozšířené notifikace, Google Now, Google Wallet a vylepšené hlasové vyhledávání.

KitKat 4.4 - Google oznámil vývoj 3. září 2013. Tato verze poskytuje bezdrátový tisk, zabudovaný nahrávání obrazovky, rozšířenou funkcionalitu pro notifikace, Bluetooth Message Access Profile a mnoho dalšího.

4.4 Použité nástroje pro vývoj

Zde se nachází základní specifikace nástrojů či prostředků, které se používají v aplikaci této diplomové práce nebo slouží pro vývoj aplikace jako takové.

4.4.1 Vývojové prostředí

V současné době jsou některá vývojová prostředí ještě v beta verzích a neumožňují plnou funkcionalitu. Například Android Studio se zařazuje stále do těch nástrojů, které ještě neobsahují veškerou funkcionalitu, příkladem toho je podpora JNI.

Eclipse

Vývojové prostředí - IDE (Integrated Development Enviroment), které je převážně napsané v programovacím jazyce Java [9]. Využívá se k další tvorbě programů. Ty se píší v různých programovacích jazycích jako například C, C++, PHP, Python, Perl a mnoho dalších. Můžeme ho zároveň použít pro vytváření balíčků pro program Mathematica. Zahrnuje v sobě nástroje (JDT) pro Javu, Eclipse CDT pro C/C++ a Eclipse PDT pro PHP. Abychom mohli vyvíjet pro mobilní platformu Android, je nutné doinstalovat ADT Plugin. Zároveň je nutné zmínit, že Eclipse není závislá na platformě. Prozatím se považuje za oficiální nástroj pro vývoj aplikaci pro Android OS.

Android Studio

Nové vývojové prostředí založené na IntelliJ IDEA [14]. Je podobný Eclipse s ADT Pluginem. Poskytuje zabudované vývojové nástroje pro vyvíjení a ladění. Nabízí nám zároveň nástroje jako překládání pomocí Grandle, Android-specifické nahrazování, rychlé opravy, ProGuard, podepisování aplikací a podporu pro Google Cloud Platform. V současné době je k dispozici pouze Preview verze.

4.4.2 Nástroje

Text-To-Speech

Umožňuje syntézu řeči, což je umělá tvorba lidské řeči tzv. provádí převod textu do mluvené podoby [2]. Tato funkcionalita se dá využít prostřednictvím android.speech.tts a přidala se do API úrovně 4. Zároveň je nutné podotknout, že nepodporuje všechny světové jazyky. To se stává potřebným pro výslovnost některých slov, protože například Paříž zní jinak v angličtině a jinak ve francouzštině. V tomto díle se používá TTS pro přečtení ohmické hodnoty rezistoru.

Camera API

Android Framework zahrnuje podporu možnosti využití fotoaparátu z různých druhů mobilních zařízeních, který umožňuje vytvoření snímků a videosekvencí [10]. Pro to, abychom mohli získávat snímky z kamery, tak se musí nastavit povolení v AndroidManifestu. Zároveň je možné využít již stávající aplikace fotoaparátu nebo si lze napsat svou vlastní aplikaci. Dále se zde může nastavit nástroje jako je blesk, GPS data, zaostření, vyvážení bílé, barevné efekty nebo detekce obličejů.

OpenCV

Umožňuje digitální zpracování obrazu a díky své optimalizaci umožňuje využití všech procesorů, a tím můžeme dosáhnout zpracování obrazu real-time. Funguje pod licencí BSD (svobodná pro akademické i komerční použití). Zahrnuje v sobě rozhraní pro C++, C, Python a Javu a je multiplatformní.

Aplikace vytvořená v této diplomové práci funguje z velké části právě na této knihovně. Pro správný běh je nutné jako první nainstalovat SDK. Softwarový vývojový kit OpenCV obsahuje mimo knihovny i ukázky aplikací demonstrující základní práci s prvky obsaženými v knihovně. V mobilním zařízením, který funguje na operačním systému Android, je následně potřeba nainstalovat program zvaný OpenCV Manager. Díky nástroji OpenCV se práce v oblasti zpracování obrazu znatelně zrychlí než kdyby se veškeré metody a funkce implementovaly v Javě. Více informací o této knihovně můžeme nalézt v [24].

5 Obrazové zpracování

V současné době, kdy moderní technika stále více zasahuje a usnadňuje denní život, je digitální zpracování obrazu neodmyslitelnou součástí rozvoje. Tato disciplína nám umožňuje detekovat obličeje na fotografii, provést různé filtry, které se aplikují na snímek pro estetičtější efekt nebo nástroje, jež umožní korekci obrazu pro odstranění nejrůznějších nedokonalostí v obraze.

Obraz můžeme chápat jako dvourozměrné pole o souřadnicích x, y. Každý takto zaměřený bod se nazývá pixel, z kterého můžeme získat vlastnosti jako je například alfa kanál a barevnou složku. Nemusí tomu být vždy tak, závisí na vlastnostech obrazu, protože ten může být černobílý a reprezentace pixelu potom bude intenzita. V reálném životě tvoří barvu světlo o určité vlnové délce.

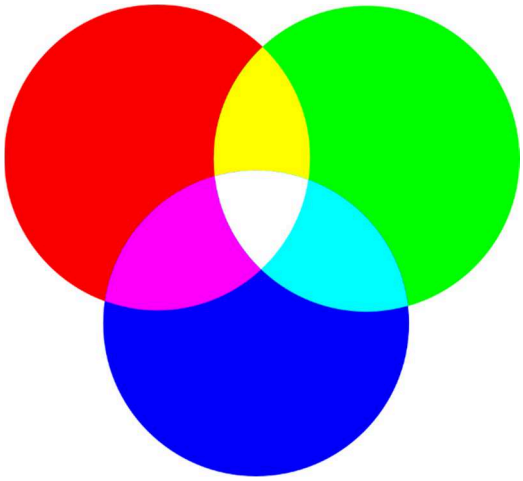
V mém případě budeme potřebovat algoritmy pro získání zájmové oblasti, tzv. musíme udělat výřez tak, abychom obdrželi rezistor z obrazu pro pozdější zpracování.

5.1 Barevné modely

Jedná se o matematickou reprezentaci sady barev. Mezi nejpopulárnější barevné prostory se můžou řadit RGB (používá se v počítačové grafice); YIQ, YUV, YCbCr (používané ve video systémech) a CMYK, který se používá při tisku [19]. Nicméně žádný z těchto prostorů není přímo spojený s odstínem, sytostí či jasem. To mělo za následek to, že se vyhledávaly další modely, které by toto splňovaly. Výsledkem tohoto hledání byly prostory HSI a HSV, které se snaží zjednodušit programování, zpracování a manipulaci koncového uživatele. Všechny barevné prostory lze odvodit z informací RGB poskytovanými zařízeními, jako jsou fotoaparáty nebo skenery.

5.1.1 **RGB**

RGB barevný model je rozšířen v počítačové grafice [19]. Model obsahuje základní tři barvy – červenou, zelenou a modrou. Společně se řadí do aditivního míchání barev, protože pokud se sčítají, vytváří nám světlo jiné vlnové délky, a tím získáme barvu jinou. Například smícháním červené a zelené získáme žlutou. Toto skládání popisuje (Obr. 5.1). V počítačové grafice je rozšířen proto, že většina zařízení pracuje právě s těmito třemi složkami. Jejich reprezentace je typická 256 úrovněmi. Avšak v reálném světě obrazů není příliš efektivní. Důvodem je například neefektivnost při obrazovém zpracováním (složitost provést změnu intenzity pixelu), potom například lidské oko vnímá okolní svět v jiném barevném modelu. Právě z tohoto a mnoho dalších důvodů video systémy využívají jiný barevný model – YUV, YIQ a YCbCr.



Obr. 5.1: Adiktivní míchání

5.1.2 YUV

Tento model se používá v PAL a NTSC televizních standardech [19]. Y v tomto případě značí jasovou složku a UV chrominanci – barevnou složku. Vytvořil se v době, kdy bylo nutné vytvořit model, který by přenášel barevný obraz a byl kompatibilní s černobílým vysíláním. Převodní vzorce z gama upravené RGB na YUV a opačně je zobrazeno níže (5.1) a (5.2).

$$Y = 0{,}299R' + 0{,}587G' + 0{,}114B' \tag{5.1}$$

$$U = -0{,}147R' - 0{,}289G' + 0{,}436B'$$

$$V = 0{,}615R' - 0{,}515G' - 0{,}200B'$$

$$R' = Y + 1{,}140V \tag{5.2}$$

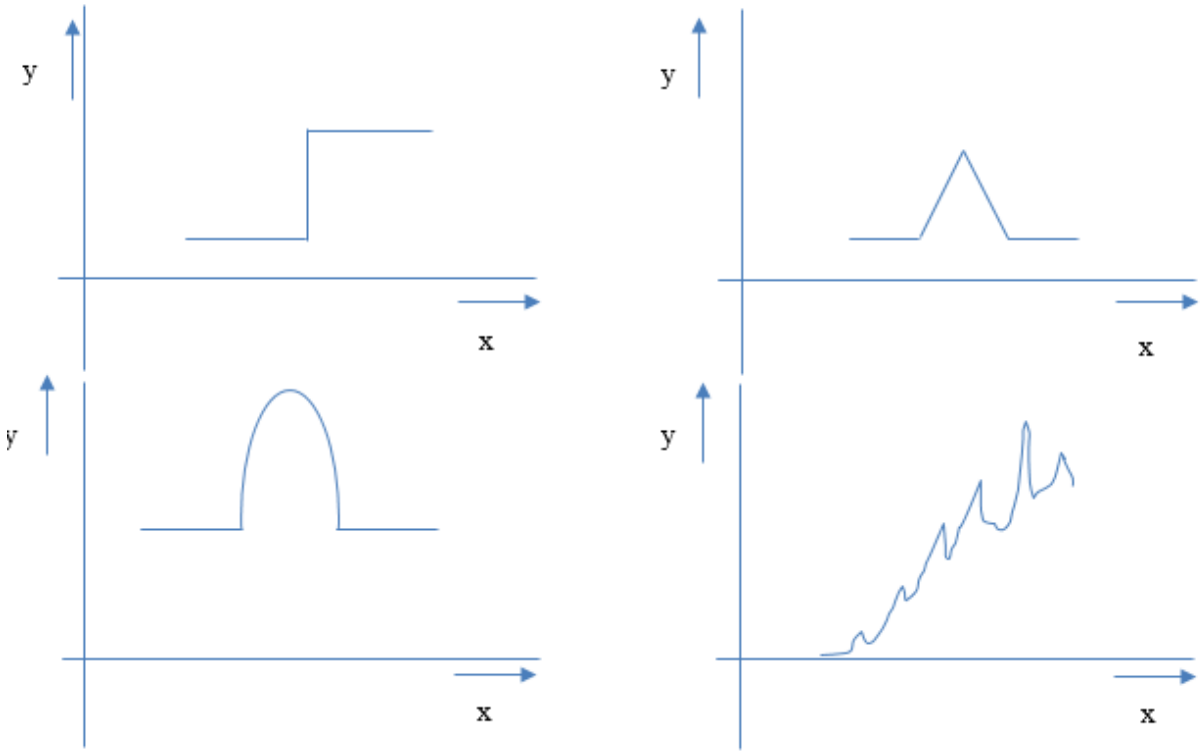
$$G' = Y - 0{,}395U - 0{,}581V$$

$$B' = Y + 2{,}032U$$

Jasová složka Y nabývá hodnot 0-255 tzv. má 256 úrovní, U rozsahu od 0 do ±112 a V od 0 do ±157.

5.2 Hranové detektory

Hrana je vlastnost objektu a jeho okolí v obraze [16]. Můžeme říci, že popisuje rychlost změny a směr největšího růstu obrazové funkce f(x, y). Lze ji tedy definovat jako velkou změnu jasu v daném místě. V reálném světě se setkáváme s hranami, které jsou v obrazovém signálu většinou zašuměné. Na (Obr. 5.2) můžeme vidět jak ideální, tak reálnou reprezentaci hrany signálem.



Obr. 5.2: Zleva doprava, z vrchu dolů, skoková, střechová, liniová a zašuměná hrana

Úkolem detekce hran je v podstatě nalezení množiny pixelů takových, kde se mění jas [16]. Hrany mohou vznikat, ale i zanikat na úhlu pohledu. Skokovou hranu můžeme najít na rozhraní světla a stínu, střechovou hranu zase u trojúhelníkových objektů. Pokud tedy hranu definujeme jako velkou změnu jasové funkce, pak se bude vyskytovat v místě hrany velká hodnota derivace jasové funkce. Při vyhledávání hran můžeme použít konvoluční masky, někdy taktéž nazývané jako konvoluční jádro, například o velikosti 3x3. Pak taková maska vypadá jako dvourozměrné pole o rozměru 3x3.

Gradientní operátory můžeme rozdělit do tří skupin [16]. Operátory aproximující derivace pomocí diferencí obsahují některé operátory, které jsou invariantní vůči rotaci, např. Laplacián a dají se počítat konvolucí s jedinou maskou. Jiné potřebují několik masek, které odpovídají příslušné orientaci. Z nich se potom vybere právě ta, které nejlépe aproximuje obrazovou funkci. Další operátory jsou založené na hledání v oblastech, kde druhá derivace obrazové funkce prochází nulou. Příkladem těchto hranových detektorů je Cannyho hranový detektor. Do poslední skupiny se řadí operátory snažící se lokálně aproximovat obrazovou funkci poměrně jednoduchým způsobem, většinou dvěma proměnnými nebo polynomem.

Robertsův operátor

Jedná se o nejstarší a zároveň nejjednodušší operátor (5.3). Jeho hlavní nevýhodou je velká citlivost na šum, protože okolí pro aproximaci je příliš malé, tzv. využívá jen okolí o 2x2 reprezentativního pixelu.

$$h_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, h_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{5.3}$$

Laplaceův operátor

Tento gradientní operátor (5.4) je poměrně rozšířený a oblíbený z důvodu jeho vlastností. Aproximuje druhou derivací, je invariantní vůči otočení a udává velikost hrany a ne její směr. Bývá také aproximován diskrétní konvolucí. Ukázku jeho konvolučních masek můžeme nalézt níže (4-sousedství a 8-sousedství).

$$h_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, h_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{5.4}$$

Sobelův operátor

Podobá se operátoru od Prewittové s tím rozdílem, že přidává vyšší důležitost prostřední buňce (5.5). Využívá se ve velké míře pro svislou a vodorovnou detekci. Mezi jeho nevýhody se taktéž považuje malá velikost konvoluční masky a pro dosažení lepších výsledků je nutné použít větší matice. Obraz se transformuje tak, že místa s měnícím se jasem označí bíle a oblasti s konstantním jasem černě.

$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, h_2 = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{5.5}$$

Problém výše popsaných hranových detektorů je velká závislost jejich chování na konkrétním obrázku, protože velikost konvolučního jádra závisí na detailech v dané fotografii. Dalším značným problémem se stává šum nacházející se v obraze.

Proto koncem sedmdesátých let vznikl projekt, který formuloval Marrovu teorii. Ten by popisoval matematický model detekce skokových hran. Základem je model, který usiluje o hledání polohy hrany ve fotografii v místě průchodu druhé derivace obrazové funkce nulou [16]. První derivace totiž nabývá svého maxima v místě hrany. Druhá derivace prochází v okolí hrany nulou.

Cannyho hranový detektor

Základní myšlenkou detektoru je představa, která říká, že skokovou hranu lze hledat filtrem [17]. Vychází z jiné motivace než Marrova teorie i když se tato úloha řešila na stejném pracovišti asi o čtyři roky později. Úloha byla předána doktorandovi J. Cannymu, který tvrdil, že detektor pro skokové hrany se stává optimálním, pokud splní tři požadavky. Detekční kritérium určuje, aby se nepřehlédly významné hrany a aby se na žádnou z hran nevyskytovaly vícenásobné odezvy. Lokalizační kritérium stanovuje, že rozdíl mezi skutečnou a nalezenou hranou byl minimální rozdíl. Požadavek jedné odezvy vyžaduje, aby detektor nereagoval na jednu hranu v obraze vícenásobně. Na rozdíl od prvního kritéria, kde již tento požadavek byl určen, se zaměřuje na hrany obsahující šum a nehladké hrany, což první požadavek není schopen zajistit.

V první fázi vývoje byl hranový detektor určen pro 1D signál a musela se splňovat první dvě kritéria. Aby se dosáhlo třetího kritéria, tak se filtr numericky optimalizoval. Dále se detektor zobecnil pro 2D. Hrana se zde uvádí s polohou, orientací a velikostí. Lze dokázat, že konvoluce s dvourozměrným Gaussiánem a derivací ve směru gradientu vytváří jednoduchý, ale účinný diferenciální operátor [17]. Ten oproti Marra obsahuje i orientaci hrany. Běžným problémem v obraze je šum, který produkuje vícenásobné odezvy na hranu. Problém můžeme odstranit prahováním

s hysterezí. Pro správný výsledek záleží, aby se správně zvolilo měřítko pro operátor v závislosti na velikosti objektů v obraze.

5.3 Rohové detektory

Rohy v obraze představují důležitý konstrukční prvek a jsou proto užitečné v celé řadě aplikací [8]. Rohové body se stávají důležitými nejen v lidském vidění, kde nám dva body vytvoří hranu, ale i ve strojovém vnímání.

Přesto, že naším vnímáním rozpoznáme roh jednoduše, tak při detekci rohů automatizovaně narážíme na nejednu překážku. Správný detektor musí splňovat řadu kritérií, jako například rozpoznání mezi skutečnými a náhodnými rohy, spolehlivě detekovat rohy v reálném světě, kde se vyskytuje šum a přesně určit polohu, kde se nachází. Roh je definován jako oblast, která vykazuje silnou hodnotu gradientu ve více směrech.

Harrisův rohový detektor

Operátor, který byl vynalezen Harrisem a Stephensem [8]. Zařazuje se do skupiny, která se zakládá na předpokladu, že rohový bod existuje. Detektor by měl být izotropní, tzv. nezávislý na orientaci. Konstatuje, že místa, kde je sklon silný pouze v jednom směru, by se neměla považovat za rohy.

Výpočty se zakládají na první parciální derivaci funkce obrazu I(u, v) v horizontálním a vertikálním směru. Pro každou lokaci v obraze (u, v) vypočítáme tři hodnoty A(u, v), B(u, v) a C(u, v). Ty získáme pomocí vzorců (5.6).

$$\begin{aligned} A(u,v) &= I_x^2(u,v) \\ B(u,v) &= I_y^2(u,v) \\ C(u,v) &= I_x(u,v) \cdot I_y(u,v) \end{aligned} \tag{5.6}$$

Tyto hodnoty jsou pak součástí matice (5.7), která se nachází níže.

$$M = \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} = \begin{pmatrix} A & C \\ C & B \end{pmatrix} \tag{5.7}$$

Dále je tato matice vyhlazena pomocí Gaussovského filtru $H^{G,\sigma}$. Z toho dostáváme vztah (5.8).

$$\bar{M} = \begin{pmatrix} A * H^{G,\sigma} & C * H^{G,\sigma} \\ C * H^{G,\sigma} & B * H^{G,\sigma} \end{pmatrix} = \begin{pmatrix} \bar{A} & \bar{C} \\ \bar{C} & \bar{B} \end{pmatrix} \tag{5.8}$$

Pokud je matice diagonální, tak můžeme nahradit hodnoty A a B vlastními čísly λ viz (5.9).

$$\begin{aligned} \lambda_{1,2} &= \frac{stopa(\bar{M})^2}{2} \pm \sqrt{\left(\frac{stopa(\bar{M})}{2}\right)^2 - \det(\overline{M})} = \\ &= \frac{1}{2}\left(\bar{A} + \bar{B} \pm \sqrt{\bar{A}^2 - 2\bar{A}\bar{B} + \bar{B}^2 + 4\bar{C}^2}\right) \end{aligned} \tag{5.9}$$

Tato vlastní čísla, které jsou pozitivní, obsahují důležité informace o struktuře obrazu. Dá se říci, že kódují sílu hrany a jejich přidružené vektory představující orientaci. Roh by měl mít silnou hranu v hlavním směru (odpovídající větší hodnotě z vlastních čísel). Jak se vypočítá rozdíl dvou vlastních čísel, můžeme vidět na vzorci (5.10)

$$\lambda_1 - \lambda_2 = \sqrt{\frac{1}{4} \cdot (stopa(\bar{M}))^2 - \det(\bar{M})} \quad (5.10)$$

Stopa značí součet všech čísel na hlavní diagonále, det je determinant a $\bar{M}$ je symetrická matice s vlastními čísly. V každém případě, kde platí $\frac{1}{4} \cdot \left(stopa(\bar{M})\right)^2 > \det(\bar{M})$ se definuje vztah (5.11)

$$Q(u,v) = \det(\bar{M}) - \alpha\left(stopa(\bar{M})\right)^2 = (\bar{A}\bar{B} - \bar{C}^2) - \alpha(\bar{A} + \bar{B})^2, \quad (5.11)$$

kde $\bar{A}, \bar{B}, \bar{C}$ znamenají funkce, na které byl aplikován Gaussovský lineární filtr a α určuje citlivost detektoru. Q(u, v) se nazývá funkce rohové odezvy a vrací maximální hodnoty v izolovaných rozích. V praxi se dosazuje za α hodnota v rozmezí od 0,04 do 0,06. Čím je tato hodnota vyšší, tím je detektor méně citlivý a detekuje se méně rohů. Poloha v obraze (u, v) se označuje za roh, pokud splňuje $Q(u, v) > t_H$ (t_H je práh) a typicky leží v mezích 10 000 a 1 000 000. Potom se takový bod vloží do pole, které obsahuje všechny rohy. Jeden takový element disponuje souřadnicemi u a v.

5.4 Další metody

V této kapitole můžeme najít další metody obrazového zpracování, které se nedají zařadit do předešlých kapitol počítačového vidění.

5.4.1 Houghova transformace

Mezi jednu z důležitých metod se považuje Houghova transformace vyvinutá Paulem Houghem a byla vydána jako patent [7]. Tento algoritmus slouží pro hledání jednoduchých obrazců v obraze, jako jsou úsečky, kružnice, elipsy, atd.

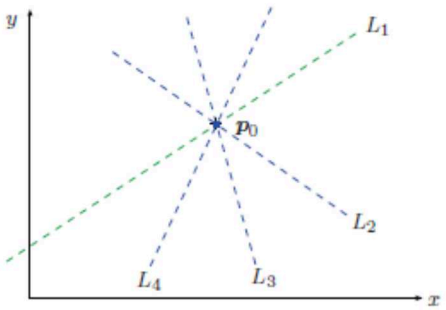
Považuje se za nejčastěji používanou metodu pro detekci úseček. Klasická úsečka se dá popsat pomocí vztahu (5.12)

$$y = kx + d, \quad (5.12)$$

kde k je směrnice a d určuje její svislý posun. Úsečka prochází dvěma body $p_1=(x_1,y_1)$ a $p_2=(x_2,y_2)$. V takovém případě platí vztah (5.13).

$$y_1 = kx_1 + d \;\; a \;\; y_2 = kx_2 + d0 \quad (5.13)$$

Cílem je najít hodnoty k a d tak, aby okrajové body ležely na trati. Houghova transformace se na to dívá z jiného pohledu. Zkoumá všechny čárové segmenty, které prochází jedním daným bodem.



Obr. 5.3: Sada čar

Převzato z: [7]

Na (Obr. 5.3) můžeme vidět sadu čar, které prochází bodem p_0 pro všechny čáry L_j. V takovém případě platí vztah (5.14).

$$L_j: y_0 = k_j x_0 + d_j \tag{5.14}$$

Vztah mezi (x, y) obrazovým prostorem a (k, d) parametry jsou znázorněny (5.15).

$$Bod: p_i = (x_i, y_i); Čára: M_i: d = -x_i k + y_i \tag{5.15}$$

$$Čára: L_j: y = k_j x + d_j; Bod: q_j = (k_j, d_j)$$

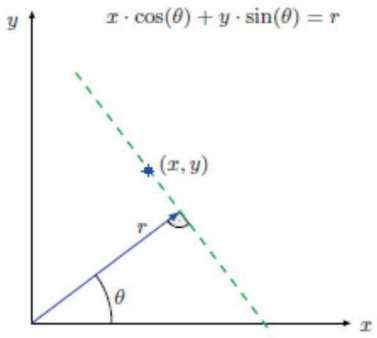
Pro každý bod p_i jsou přiřazeny úsečky M_i, v prostoru parametrů.

Nalezení dominantních úseček v obraze může být nyní přeformulováno jako nalezení oblastí, kde se protíná značný počet úseček [7]. To je v podstatě cílem Houghovy transformace. Musíme nejprve rozhodnout o diskrétní reprezentaci spojitého parametru pomocí vhodného výběru pro k a d osy. Poté, co jsme vybrali velikost kroku pro souřadnice, tak prostor můžeme reprezentovat jako dvourozměrné pole. Toto pole uchovává hodnoty prostorových parametrů protínajících se čar.

V reálném případě vzorec (5.12) nelze použít, protože pro vertikální přímky je sklon nekonečný, čili k = ∞. Praktičtější zastoupení je pomocí Hessianovy normální formy pro reprezentaci přímek viz vzorec (5.16)

$$x \cdot \cos(\theta) + y \cdot \cos(\theta) = r, \tag{5.16}$$

který nevykazuje tyto zvláštnosti a poskytuje lineární kvantizaci pro své parametry úhel θ a poloměr r viz (Obr. 5.4).



Obr. 5.4: Znázornění, jak se vyhnout problém s použitím úhlu a poloměru

Převzato z: [7]

Pokud budeme používat střed obrazu jako referenční bod pro obrazovou plochu x/y, tak pak je možné omezit rozsah poloměru na polovinu [7]. Toto tvrzení nám demonstruje vzorec (5.17), kde M je šířka a N výška.

$$-r_{max} \leq r_{x,y}(\theta) \leq r_{max}, kde\ r_{max} = \frac{1}{2}\sqrt{M^2 + N^2} \tag{5.17}$$

5.4.2 Blob detekce

Jedná se soubor matematických metod v oblasti počítačového vidění, které mají za úkol porovnávat oblasti, které se liší ve vlastnostech, jako jsou barvy nebo jas, s okolními oblastmi. Blob

lze definovat jako skupinu pixelů, která splňuje určitá kritéria. Zároveň se používá pro získání doplňujících informací, které nám neposkytují obrazové techniky jako je například detekce hran [11]. Můžeme tedy získat informace, jako jsou plocha či obvod regionu. Existují dvě hlavní třídy blob detektorů [21]. První z nich pracuje na diferenciálních metodách, které jsou založeny na derivátech funkce s ohledem na pozici. Druhá je založená na zjištění lokálních extrémech funkce. Používané operátory jsou:

- Laplacián Gaussiánu
- Bloby založené na stupních šedi
- Rozdíly Gaussovského rozostření
- Determinant Hessiánu
- Hybrid složený z Laplaciánu a determinantu Hessiánu

6 Struktura aplikace

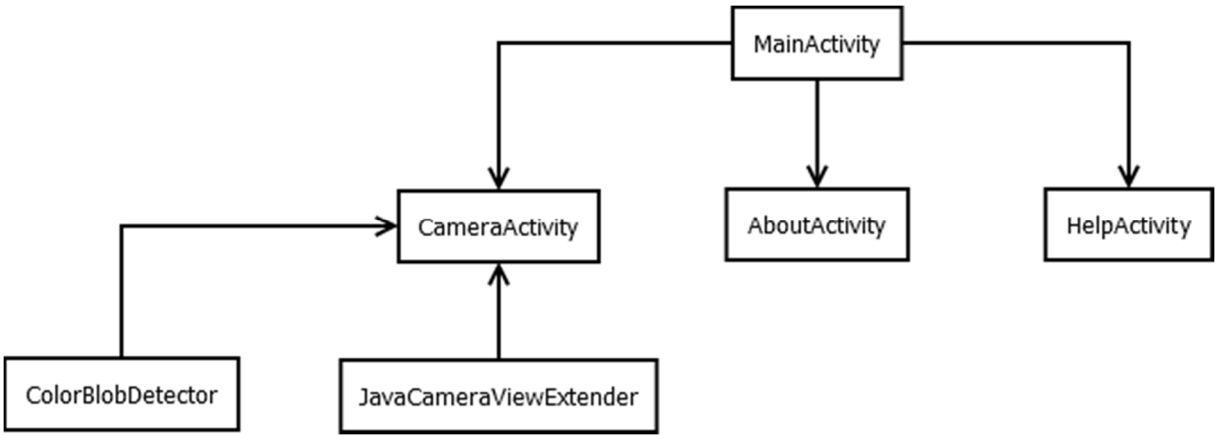
Na úvod bych rád řekl, že pokračuji ve své bakalářské práci, která nesla název Čtečka barevných kódů rezistorů pro Android. Některé věci, jako pozadí a ikony jsou převzaté z této práce. Tato aplikace fungovala tak, že se vyfotografoval rezistor v horizontálním směru a umístěním tak, že pruh tolerance byl na pravé straně. Odchylky v úhlu se neuvažovaly, protože pro detekci byl použit Sobelův operátor, který není invariantní vůči rotaci a tak konvoluční jádra měla přesný tvar pro správnou detekci. Navíc uživatel musel vyříznout obraz, který sloužil jako oblast zájmu. Tyto a další jiné aspekty vedly autora k zamyšlení, aby využil vhodnějších postupů a algoritmů, které by přispěly ke komfortnějšímu používání aplikace a aby výsledky aplikace byly lepší než v případě bakalářské práce.

Obrázek (Obr. 6.1) představuje třídní diagram, kde můžeme nalézt jednotlivé vlastnosti a metody daných tříd. Základní třídou je MainActivity, která umožní uživateli se dostat do jiných částí aplikace. Možnost, kam se uživatel může dostat je, že buď spustí aktivitu CameraActivity, AboutActivity nebo HelpActivity. První ze jmenovaných obsahuje celou logiku obrazového zpracování. Na ni se vážou další třídy jako JavaCameraViewExtender a ColorBlobDetector.

JavaCameraViewExtender slouží pro správu a obsluhu fotoaparátu a lze zde nastavit věci jako rozlišení, režim zaostřování nebo režim blesku. Jedná se o potomka třídy JavaCameraView, která je naimplementována v OpenCV a zároveň je potomkem třídy CameraBridgeViewBase, kterou lze najít rovněž v OpenCV a dědí ze třídy SurfaceView. Tuto třídu lze již běžně používat a rozšiřovat, protože je obsažena přímo v Androidu.

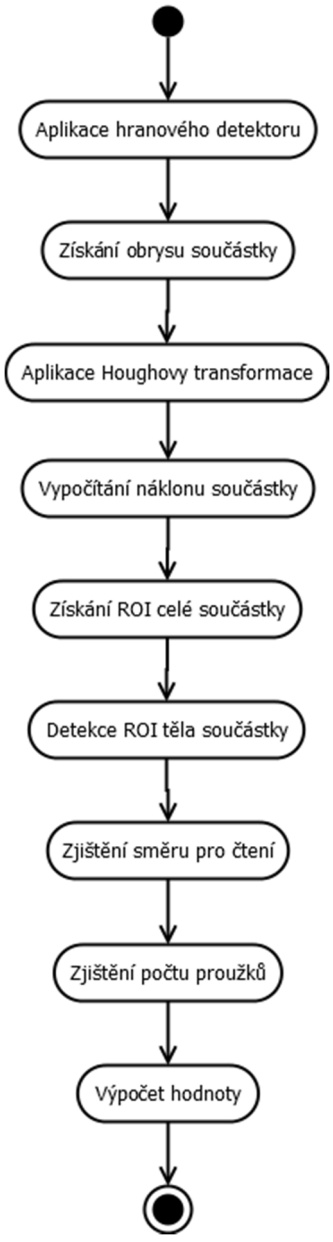
ColorBlobDetector je třída, která pomáhá vyhledávat konkrétní shluk barev v obraze. Tuto třídu lze najít v ukázkách, které se instalují společně s SDK pro OpenCV. Pracuje s barevným prostorem RGB, ten se převádí na HSV, kde se vyhodnocuje konkrétní blok barev.

Aktivita AboutActivity poskytuje informace o autorovi. HelpActivity nabízí uživateli doporučení a návody, jak správně aplikaci používat. Pouze při dodržení těchto podmínek se dosáhne optimálního vyhodnocení.



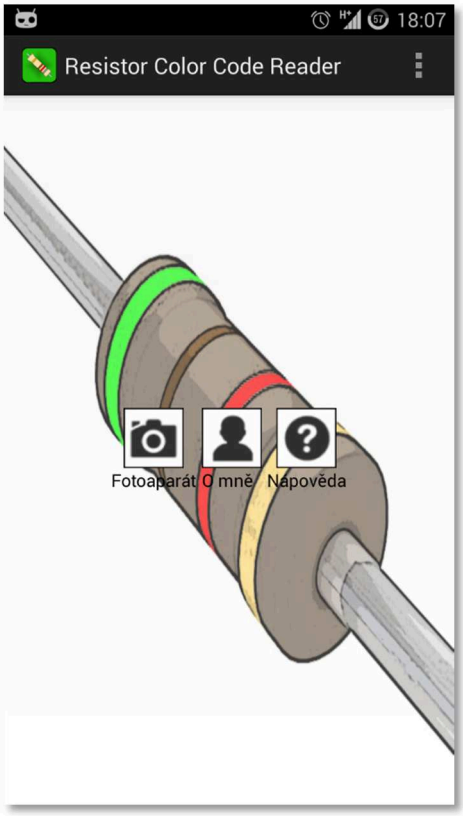
Obr. 6.1: Zjednodušený třídní diagram

Na (Obr. 6.2) vidíme diagram aktivit, který popisuje sled událostí od nalezení rezistoru v obraze až po zjištění jeho výsledné hodnoty. V prvním kroku je důležité v obraze použít vhodný hranový detektor, který nám následně poskytne obrysy, jež nám udávají umístění součástky v obraze. Nadále se musí aplikovat Houghova transformace pro vyhledávání přímek v obraze. Ta nám nalezne přímku v obraze, kde se rezistor nachází, a z této přímky jsme schopni vypočítat úhel naklonění součástky. Následně zjistíme, kde se v oblasti zájmu nachází tělo rezistoru. Potom nám zbývá správně klasifikovat barevné pruhy, zjistit jejich počet a vrátit vypočítaný výsledek.



Obr. 6.2: Diagram aktivit

Samozřejmě se jedná o zjednodušený model, který nám říká, jak řešit daný problém. Konkrétní kroky vypracování lze vyčíst z kapitoly 7 Implementace nebo popřípadě ve zdrojových kódech, které jsou součástí přiloženého CD. Diagram aktivit nám umožní stanovit cíle, kterých je potřeba dosáhnout. Při splnění těchto podmínek můžeme získat aplikaci, která je schopná vypočítat hodnoty rezistorů.



Obr. 6.3: Screenshot hlavního okna aplikace

Na (Obr. 6.3) můžeme vidět hlavní okno aplikace. Jak bylo uvedeno výše, tak se uživatel může dostat do dílčích aktivit. Grafický návrh, podoba ikonky a název aplikace se převzal z mé bakalářské práce Čtečka barevných kódů rezistorů pro Android. Jediné, co se změnilo, je počet ikonek. Toto se děje z důvodů převedení aplikace do real-time podoby. Tímto nám odpadají kroky spojené se získáváním fotografie, protože původní aplikace zpracovávala obraz až po jejím získání. Ikonky v aplikaci byly převzaté z balíčku od společnosti Google Inc. a byly zvoleny tak, aby měly vypovídající hodnotu. Pozadí se upravilo v grafickém editoru Photoshop tak, že se na něj aplikovala transformace a následně filtr Plakátové obrysy.

7 Implementace

Implementace se prováděla v programovacím jazyce Java s využitím knihovny pro obrazové zpracování. Jako vývojové prostředí bylo použito Eclipse, protože Android Studio je ještě v beta verzi a neumožňuje některé funkcionality. První jmenovaný spadá pod licenci Eclipse Public License a druhý pod Apache 2.0. Jako testovací přístroj se využil Samsung Galaxy SIII, který by měl poskytnout dostatek výpočetního výkonu a zároveň disponuje přisvětlovací diodou, která může zajistit konstantní osvětlení. Zároveň se musely testovat dané implementace při určité světelnosti a k tomu nám pomohl luxmetr UA1010B, který disponoval 3 ½ místným displejem, rozsahem 200 – 200000 lx a chybovost se uvádí kolem ± 3 % z přečtené hodnoty + 0,5% z rozsahu.

7.1 Nastavení

Nastavení fotoaparátu bylo takové, že pro průběžné zaostřování objektů se použil mód FOCUS_MODE_CONTINUOUS_PICTURE, který nejvíce vyhovoval mé aplikaci. Ostatní módy, jako makro mód se zaostřil pouze jednou a při pohybu mobilním telefonem došlo k rozostření. Při použití FOCUS_MODE_CONTINUOUS_VIDEO zaostřování nebylo tak agresivní jako u prvně zmiňovaného módu, který je k dispozici až od API verze 14. Pro konstantní osvětlení se zkusil použít režim FLASH_MODE_TORCH. Ten zajistí, že při spuštění třídy, kde se vykonává obrazové zpracování, se spustí přisvětlovací dioda. Její barva světla se přibližuje dennímu, což odpovídá 5000 K – 7000 K a přispívá intenzitou osvětlení přibližně 770 lx. Měření intenzity osvětlení přisvětlovací diody proběhlo ve vzdálenosti zhruba 20 cm. Základní rozlišení fotoaparátu jsme použili 8 megapixelů. A oddálení od elektronické součástky činilo přibližně 20 cm a směr náklonu mobilního zařízení by mělo být kolmé. Orientace obrazovky je nastavena na hodnotu landscape. Pro správné fungování aplikace je nutné si stáhnout z Google Play program OpenCV Manager, který nám zajistí podporu knihovny pro obrazové zpracování.

7.2 Použité prostředky

Pro rychlejší fungování a digitální zpracování obrazu se použila knihovna OpenCV, která v sobě již obsahuje metody, jež se dají rovnou volat. Není tedy potřeba, aby se implementovaly. Verze knihovny, která se používá v této diplomové práci, je 2.4.8.

Dále se používá Text-to-speech, který je přímo naimplementován v Androidu. Tato funkcionalita slouží pro hlasovou reprezentaci výsledku. Základní metody programování pro Android můžeme nalézt v [4] a [22].

7.3 Hranový detektor

Jako hranový detektor se zvolil Cannyho hranový detektor pro jeho lepší výsledky při různých podmínkách osvětlení a lepší odezvu na barvy. Při použití Sobelova hranového detektoru vznikl problém, že při detekci hrany nám vznikl gradient v tomto místě. V obraze nám tedy vznikla množina čar, které indikovaly hranu. Další nevýhodou se stalo konvoluční jádro, které se musí vhodně zvolit. Pokud se tedy v aplikaci použije příliš vysoké rozlišení, tak matice musí mít odpovídající velikost.

V případě použití masky o velikosti 3x3 se některé barvy vůbec jako hrana neidentifikovaly a i malý šum v obraze znamenal zhoršení výstupu. Navíc je nutné sledovat směr, na který má hranový detektor reagovat. Na (Obr. 7.4) je možné vidět porovnání Cannyho a Sobelova detektoru při stejných podmínkách a intenzitou osvětlení 746 lx. Bez dalších úprav obrazu jsou výsledky Sobelova detektoru daleko horší než Cannyho. Proto pro lepší výstup se před aplikací masky musel obraz rozostřit (např. Gaussovo rozostření).

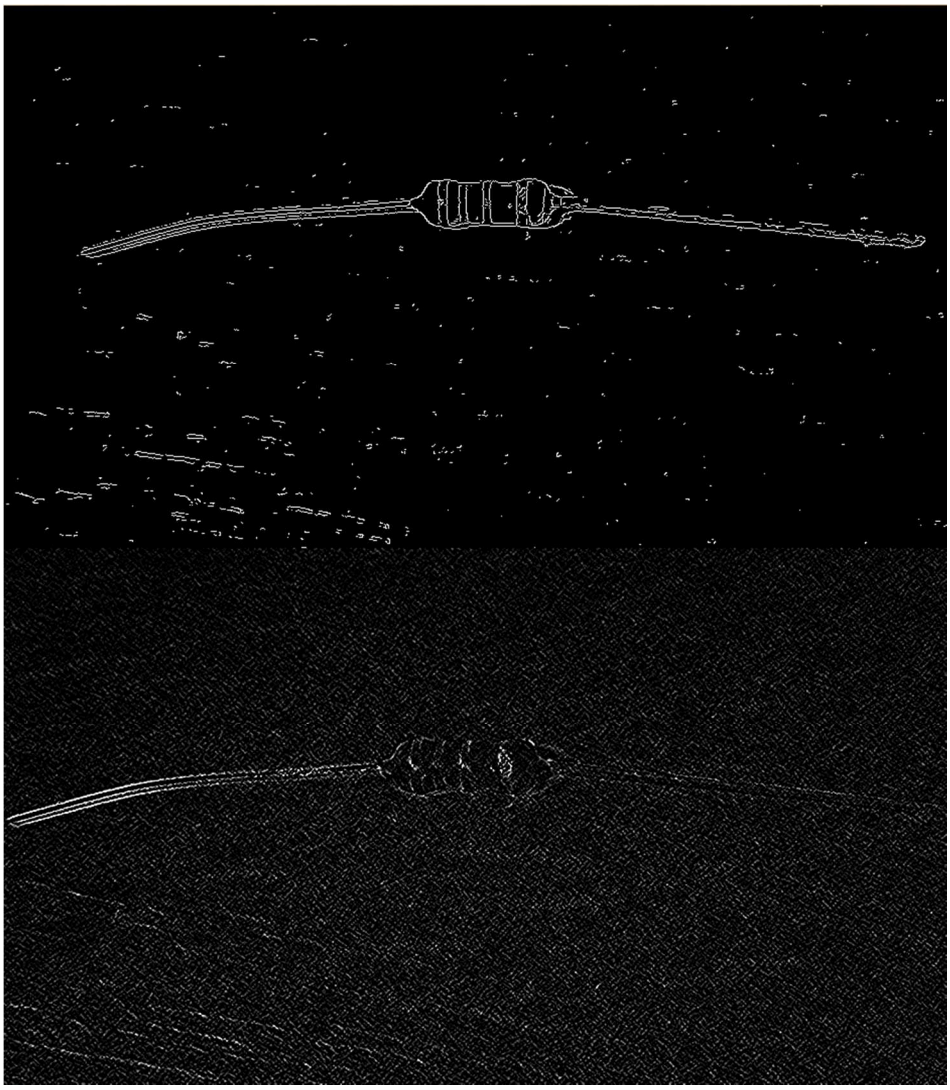
V OpenCV se volá funkce ve tvaru:

Imgproc.Sobel(grayInnerWindow, mIntermediateMat, CvType.CV_8U, 1, 1),

kde vstupní obraz se převádí do stupni šedi. Třetí parametr značí hloubku výstupu. Poslední dvě čísla jsou hodnoty pro derivaci dx a dy. Velikost jádra se stanovila na 3x3. U Cannyho má metoda tvar

Imgproc.Canny(rgbaInnerWindow, mIntermediateMat, 80, 100),

kde vstup je v RGB barevném modelu a poslední dvě hodnoty nastavují práh pro hysterezní funkci.



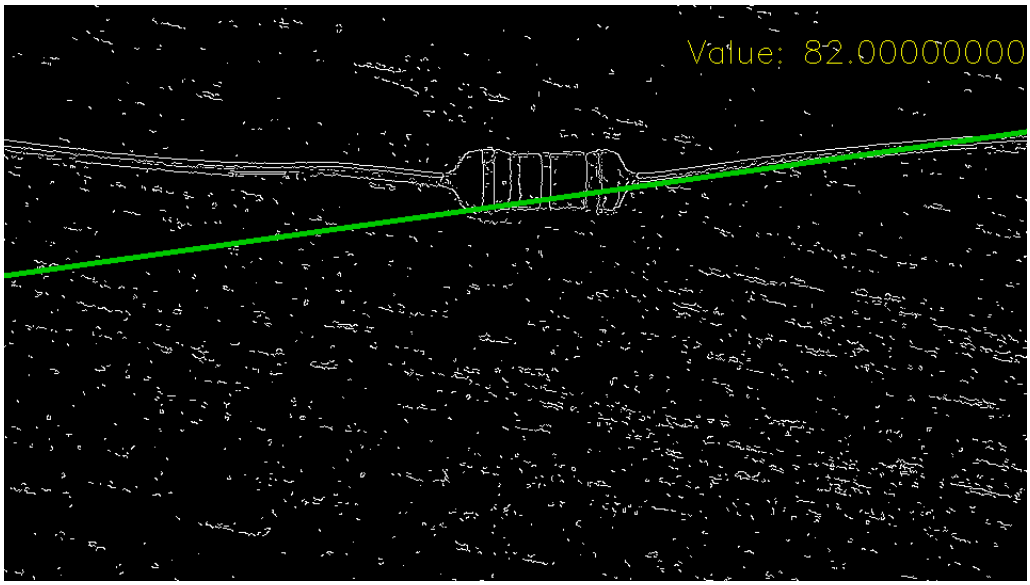
Obr. 7.4: Porovnání hranových detektorů, nahoře Cannyho, dole Sobelův

7.4 Rohové detektory

V práci se vyzkoušel použít Harrisův rohový detektor, který bohužel při detekci jako takové vykazoval zhoršené výsledky. Hlavní příčinou špatné detekce bylo to, že tělo rezistoru je zakulacené, proto algoritmus nebyl schopný přesně najít hrany součástky.

7.5 Houghova transformace

Na (Obr. 7.5) můžeme vidět příklad modifikace Houghovy transformace pro vyhledávání přímek v obraze. Tato přímka je nám schopná dodat informace jako je úhel natočení součástky. Podmínkou pro správnou detekci přímky je, aby rezistor měl své nožičky narovnané (nemusí být úplně narovnané, ale nesmí být zahnuté).



Obr. 7.5: Ukázka detekce přímky v obraze a vyhodnocení úhlu náklonu

Při snížení parametru treshold ve funkci pro Houghovu transformaci lze docílit toho, že lze mít nožičky ustřižené a mírně zahnuté. V případě, že rezistor je nový a nožičky má pouze zahnuté, tak algoritmus může špatně detekovat úhel součástky, protože délka nožiček může být větší, a tak klasifikuje úhel náklonu nožičky rezistoru místo těla.

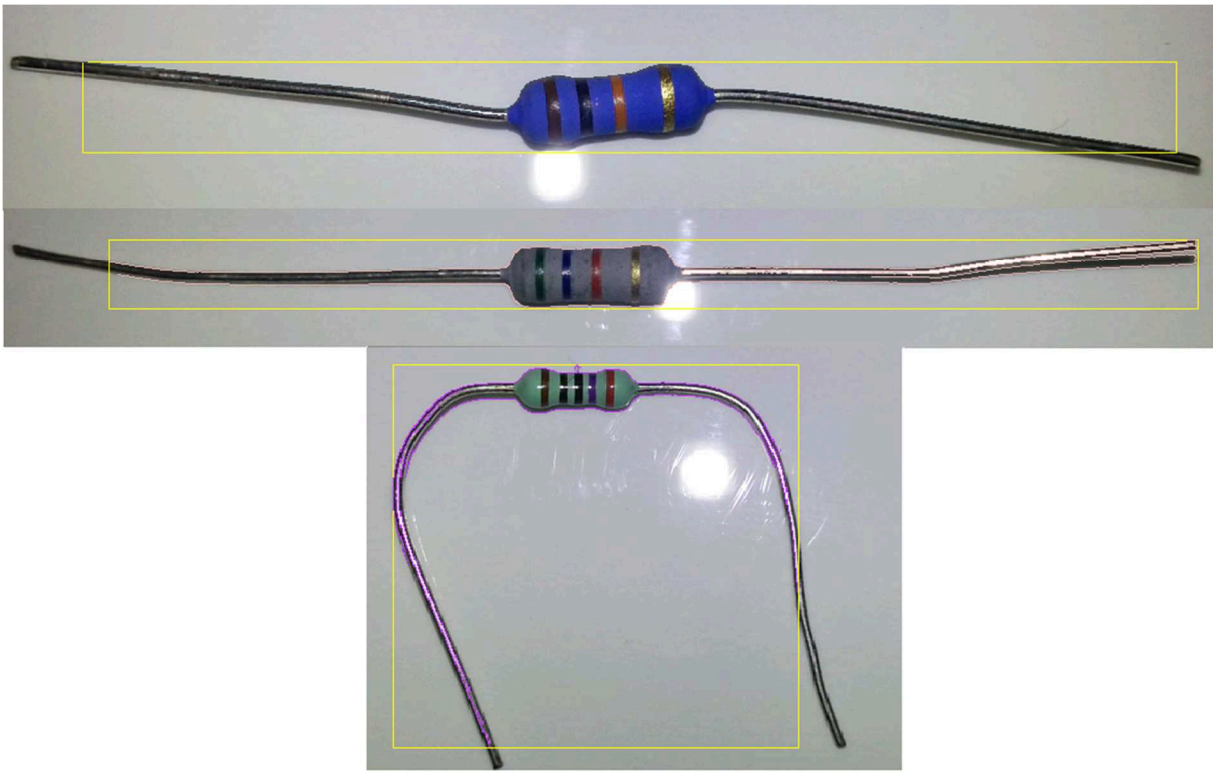
Imgproc.HoughLines(mIntermediateMat, mLines, 1, Math.PI/180, 50)

Na příkladu výše můžeme vidět ukázku funkce, která se používá v aplikaci. První parametr je zdrojová matice, mLines značí matici, kam se ukládají přímky, třetí hodnota nastavuje ρ, čtvrtý parametr nastavuje θ a poslední z nich určuje práh.

7.6 Nalezení ROI

Na následujícím (Obr. 7.6) můžeme vidět, jak aplikace nachází oblast zájmů z obrysů součástky. Při použití přisvětlovací diody je úspěšnost nalezení vyšší než bez její aktivace. Zároveň je nutné, aby součástka byla zaostřená, to se děje pravidelně po intervalech. Dále je vhodnější použít

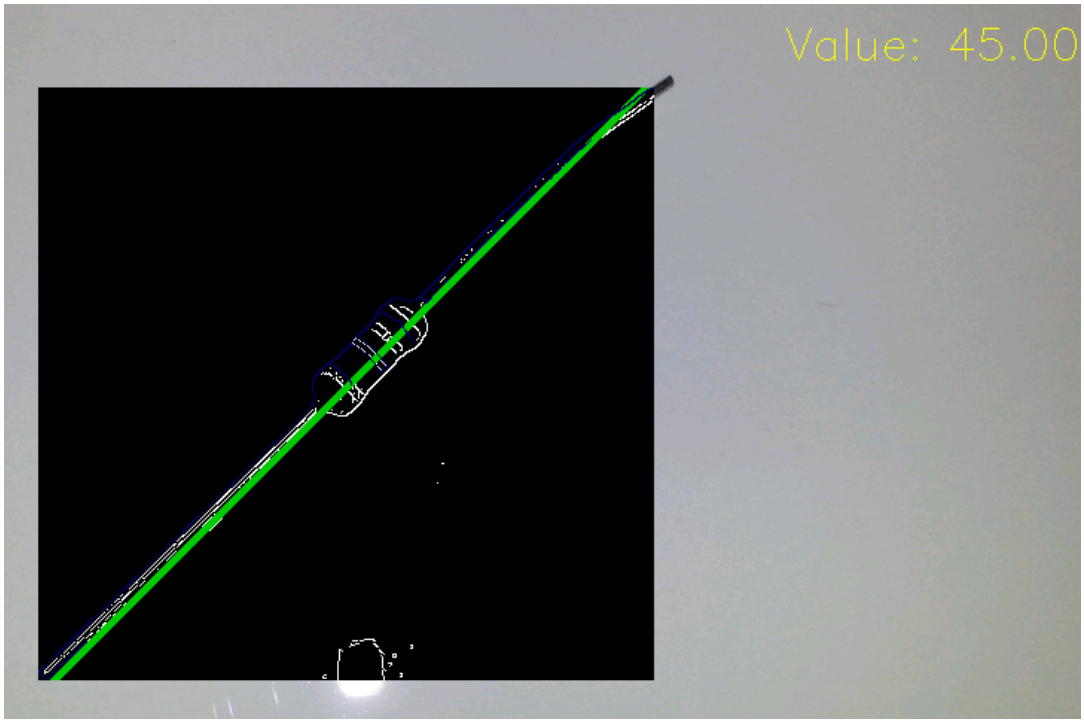
pozadí, které neodráží světlo zpět do objektivu. Za vhodné pozadí lze považovat bílý papír. Na ukázce je použit stůl, z kterého je patrný odlesk. Z obrysů se vybere ten největší pro případ, kdyby se v obraze vyskytovala například zrna prachu. Následně se kontroluje, aby plocha kontury byla větší než určitá mez. Tím ošetříme chybu, kdy algoritmus zaměří jen nožičku rezistoru nebo jen určitou část rezistoru.



Obr. 7.6: Znázornění detekce oblasti zájmu

Následně se vytvoří čtverec kolem obrysu součástky, tak získáme oblast zájmu pro celou součástku.

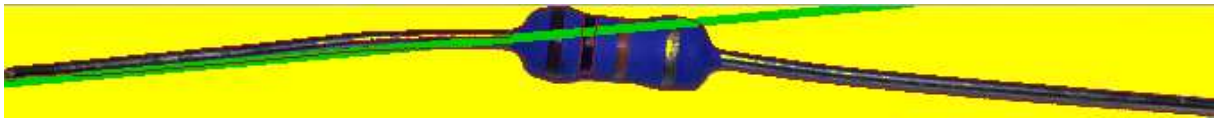
Při získání oblasti zájmu kolem celé součástky se mohla zkusit aplikovat Houghova transformace. Tato kombinace je vyobrazena na (Obr. 7.7), kde ROI se převedlo do černobílého obrazu. Nalezený úhel se demonstruje pomocí zelené přímky. Úhel 0° je detekován v případě, kdy součástka leží svisle. Tento úhel je možné dále použít pro natočení součástky tak, aby ležela horizontálně. Z této pozice se nám bude lépe zjišťovat oblast, kde se nachází tělo rezistoru. Detailně popsaná metoda pro zjištění umístění těla součástky se nachází v 7.8 Nalezení těla rezistoru. Zároveň pro správné zaostření a klasifikaci je nutné mít rezistor ve vhodné vzdálenosti od objektivu mobilního telefonu. Jak již bylo výše zmíněno, tak doporučená vzdálenost je přibližně 20 cm.



Obr. 7.7: Kombinace výběru oblasti zájmu a určení úhlu náklonu

7.7 Odstranění pozadí z ROI

Z matice, která obsahuje pouze oblast zájmu, se udělá střední hodnota barev všech pixelů, a pak se prochází obraz bod po bodu a kontroluje se, zda pixel má barevnou hodnotu vyšší než střední hodnota. Tím můžeme odstranit pozadí v matici tak, jak je zobrazeno na (Obr. 7.8).



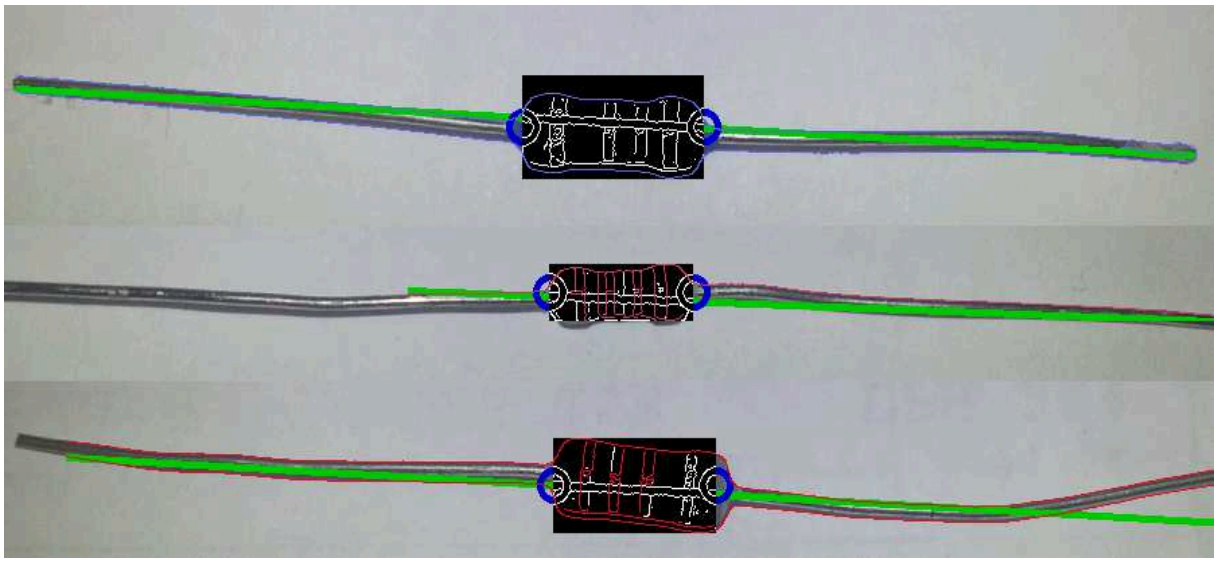
Obr. 7.8: Odstranění pozadí

7.8 Nalezení těla rezistoru

Z obrazu, který obsahuje oblast zájmu, si vytvoříme střední hodnotu. Následně si vytvoříme dvě pole čísel o velikost počtu sloupců v obraze. Jedno nám slouží jako počítadlo, kolik pixelů ve sloupci nám splňuje podmínku vyšší hodnoty v pixelu vůči střední hodnotě a druhé pole sčítá indexy řádků a na konci je zprůměruje hodnotou v počítadle. Na úvod tedy začneme procházet obraz pixel po pixelu, a pokud je hodnota v pixelu vyšší než střední hodnota vypočítaná z celého obrazu, tak se přičte do počítadla jednička, tato počítadlo se nám na začátku každého sloupce nuluje a na konci se ukládá do výše zmiňovaného pole. Na konci cyklu, který projde všechny sloupce v prvním řádku, se pak ptám, pokud je hodnota v počítadle vyšší než zadaná konstanta, tak ulož na příslušnou pozici v poli hodnotu počítadla.

Dále se vypočítá průměrná hodnota pixelů v celém obraze. Pole co obsahuje počty pixelů, které splnily podmínku, se v jednotlivých řádcích všechny hodnoty sečtou a vydělí počtem řádků,

které obsahují alespoň jeden takový obrazový bod. Po sléze vypočítáme hodnotu šířky tím, že vypočítáme souřadnice maximální a minimální hodnoty na ose x. Od maximální hodnoty odečtu minimální a dostaneme šířku. Tato šířka se musí kontrolovat, aby byla v určitých mezích. Na (Obr 7.9) můžeme vidět naznačení, kde algoritmus zjišťuje, kde se tělo součástky nachází. V případě, když je součástka nakloněná o určitý úhel, tak se vypočítá úhel pro rotaci. Ten se vypočítá tak, že vezmeme úhel 180° a od něho odečteme úhel získaný výpočetem z Houghovy transformace.



Obr. 7.9: Lokalizace hodnot na ose x pro výřez

7.9 Detekce pozadí rezistoru

Vytvoříme pole doubelů o velikosti 3. Toto pole pojmenuje backgroundColor a reprezentuje barvu pozadí. Jednotlivé hodnoty naplníme tak, že vezmeme šířku výřezu těla rezistoru a odečteme od něj konstantu. Rozdíl nám slouží jako startovací hodnota pro cyklus, ten pokračuje až po šířku výřezu. Na jednotlivé pozice v backgroundColor se ukládají jednotlivé hodnoty RGBA a podělí se rozdílem.

Nyní vytvoříme pole backgroundDistinction typu double a opět o velikosti šířky výřezu. Zároveň vytvoříme proměnnou avgDistinction typu double a přiřadíme jí hodnotu 0. Cyklus prochází pruh, který se nachází v polovině výřezu a postupně na každé pozici v backgroundDistinction uložíme rozdíl barev backgroundColor a hodnoty v daném pixelu a počet podělíme konstantou. Zároveň ukládáme do avgDistinction hodnotu na pozici indexu v poli backgroundDistinction podělenou celkovým počtem sloupců z výřezu.

Těmito dvěma způsoby získáme průměrnou barvu v celém pruhu a pole hodnot, v kterých máme uložený rozdíl barvy pozadí vůči barvě nacházejících se v jednotlivých pixelech. Podmínkou pro správné určení je, aby tělo rezistoru nebylo zanesené nečistotami nebo jinak poškozené.

7.10 Detekce barev

Pro zjištění vzdálenosti mezi dvěma barvami se používá metoda, která pracuje na principu metody vzdálenosti vektoru. První metoda je shodná jako (2.1). Druhá metoda nepočítá s odmocninou, ale počítají se pouze druhé mocniny vzdálenosti jednotlivých barevných složek RGB.

V prvním kroku vytvoříme pole integerů colorDet o velikosti šířky matice, která obsahuje výřez těla rezistoru. Následně se prochází ve vertikální polovině výřezu pruh a kontroluje se podmínka, aby hodnota v poli backgroundDistinction na daném indexu byla větší než avgDistinction. Když podmínka není splněna, tak se uloží do pole colorDet na příslušný index hodnota -1. Pokud je podmínka splněna, tak se z pruhu bere barva z jednotlivých obrazových bodů a ukládá se do pole colorInPixel typu double. Následně vytvoříme proměnnou minimum typu double a uložíme do ní vysokou konstantu. Do proměnné minimumColor typu integer uložíme hodnotu -1. Cyklus prochází všechny barvy nacházející se v tabulce referenčních barev a kontroluje se, jestli vzdálenost mezi colorInPixel a hodnotou je menší než minimum. Pokud ano, tak do proměnné minimum se uloží vzdálenost a do minimumColor se uloží index, na kterém se nachází referenční barva. Tento cyklus se provádí pro všechny barvy obsažené v referenční tabulce a zjišťuje se, u které barvy je vzdálenost nejmenší. Do pole colorDet na daném indexu se pak vloží minimumColor, což je výsledná detekovaná barva.

Barevný pruh se bere jako gradient, a proto je nutné vytvořit algoritmus, který by nám nějakým způsobem vyhodnocoval četnost jednotlivých prvků v pruhu. Nejdříve se vytvoří proměnné numContinuous, která počítá počet po sobě následujících barev jiných než -1. Proměnná numCodes počítá počet nalezených barev. Dále se vytvoří pole integerů o velikosti počtu sloupců v matici matResBodyOriginal. Pole integerů result uchovává konečné barvy. Algoritmus funguje tak, že se prochází pole colorDet a hledá 7 po sobě jdoucích barev. Ty se ukládají do pole sumCodes a ve společné podmínce se přičítá jednička k proměnné numContinuous. Pokud se vyskytuje v poli colorDet -1 a počítadlo numContinuous je menší než 7, tak se do této proměnné přiřazuje nula. V případě, že se vyskytne -1 a počítadlo uchovává hodnotu vyšší než 6, tak se vytvoří pole integerů sumc o velikosti pole s referenčními hodnotami. Do tohoto pole se na určitý index přičte jednička. V případě, že detekovaná barva není -1, tak se pak vybere barva, která má nejvyšší četnost, uloží se do pole result a do obrazu se vykreslí obdélník, který má pozadí jako detekovaná barva. Nakonec se kontroluje, jestli pole result obsahuje dostatečný počet hodnot a případně cyklus ukončí a přejde se k výpočtu. Pokud podmínka není splněna, tak se vynuluje proměnná numContinuous a znovu se zinicializuje pole sumCodes. Provedení výpočtu je popsáno v kapitole 7.12 Výpočet a korekce výsledku. Použití pomocí Blob detekce nebyla potřeba. Tato třída se použila jen pro inspiraci, že tabulka referenčních hodnot byla pro daný typ rezistorů vhodně zvolená.

7.11 Tabulka referenčních barev

Tyto tabulky se plní v aplikaci pouze jednou. Barvy se získaly pomocí programu Gimp 2.8 a nástroje Barevná pipeta. Pořídilo se 400 fotografií se všemi barevnými pruhy a následně se z jednotlivých pruhů odebíraly barvy, ty se potom zprůměrovaly. Do tabulek referenčních barev 7.1 se zapsaly výše zmíněné průměry. Fotografie se pořizovaly při intenzitě osvětlení přibližně 718 lx.

Tabulka 7.1: Tabulky referenčních barev

	R_1	G_1	B_1	R_2	G_2	B_2
Černý	20	20	20	45	45	59
Hnědý	71	53	38	100	57	61
Červený	105	31	40	157	57	71
Oranžový	160	90	50	185	83	59
Žlutý	157	123	39	199	181	59
Zelený	41	70	46	51	95	79
Modrý	40	73	86	36	58	140
Fialový	75	55	75	91	51	106
Šedý	73	65	62	86	94	112
Bílý	200	200	200	200	200	200
Zlatý	189	159	96	189	159	96
Stříbrný	145	144	148	145	144	148

7.12 Výpočet a korekce výsledku

V první řadě je nutné určit, kde se nachází toleranční pruh. Z toho dále zjistíme směr, kterým se má číst výsledná hodnota. Toleranční pruh se značí pouze osmi barvami, toto můžeme pozorovat v tabulce 1.3. Zároveň se odlišuje tak, že by mezera před ním měla být větší než u ostatních pruhů. Bohužel toto se v praxi některými výrobci nedodržuje, a tak v mém případě se toleranční pruh omezil jen na stříbrnou a zlatou barvu.

Pro určení, kde se toleranční pruh nachází, se vezme pole obsahující výsledné hodnoty jednotlivých pruhů. V poli se hledá stříbrný nebo zlatý pruh, pokud je nalezen, tak se uloží do proměnné stripeOfToleranceLocation index, kde se pruh nachází. Následně se kontroluje, jestli je umístění v poli hodnot v horní nebo dolní polovině. Pokud je v dolní, tak víme, že se hodnota má číst zprava a uspořádání prvků v poli s hodnotami se přeskládá do opačného pořadí.

Vypočet závisí na počtu pruhů na rezistoru. V případě, že máme čtyř-pruhový rezistor, tak se výpočet provede tak, že vezmeme první hodnotu, ta se následně vynásobí 10 a k ní přičteme hodnotu na druhé pozici v poli. Třetí hodnota vezme z pole pro násobitele exponent. Předchozí dvě hodnoty vynásobíme 10^x, kde x je exponent pro násobitele. Pro pěti-pruhové součástky je výpočet obdobný s tím rozdílem, že třetí hodnota je číslice, která se připočítá k dvou předchozím a teprve čtvrtá hodnota znamená násobitele. Ohmická hodnota je zobrazena a vyhlášena, až se barevné pruhy ustálí a barevné značení se šestkrát po sobě opakuje.

Motivace pro vytvoření toho algoritmu byla, že pokud aplikace špatně vyhodnotí hodnotu rezistoru, např. vypočítaná hodnota se stanovila na 4 600 ohmů, tak se výsledek podle tabulky 1.2 opraví na 4 700 ohmů, protože víme, že řada E24 takovou hodnotu neobsahuje. Algoritmus pracuje s hodnotami, které se nachází v řadě E24, poněvadž se tato řada používá nejčastěji. V praxi, kdy detekovaná barva byla příliš vzdálená, tak se výsledek chybně určoval, proto se od použití upustilo.

8 Testování a dosažené výsledky

V této kapitole se uvádí testy a postupy, jakými se algoritmy testovaly a případně jak se vyhodnocovaly dílčí úspěšnosti. Z tohoto testování následně pak vyplývá manuál, který se nachází v příloze této diplomové práce.

8.1 Pozadí

Testy se prováděly na třech odlišných pozadích. První z nich byl klasický dřevěný stůl, na jehož povrchu byla imitace menších suků. Tyto suky by nám mohly v některých případech způsobit problém, protože algoritmus vyhledává veškeré kontury a v případě, že suk by byl příliš velký tzv. větší než tělo rezistoru, tak by mohly nastat problémy s vyhodnocováním. Toto pozadí se tedy zavrhlo. Další pozadí, na kterém se testovalo, byl bílý stůl s lesklým povrchem. Světlo se odráželo zpět do objektivu, ale při testování nebyly pozorovány horší výsledky. Poslední a nejstabilnější pozadí byl bílý papír. Disponuje navíc vlastností, že světlo z přisvětlovací diody neodráží zpět.

8.2 Podmínky

Podmínky při testování byly idealizované, tzv. rezistory nejevily známky poničení (ohoření, překrytí barvy pruhu, atd.). Při testování se pod objektiv fotoaparátu vkládala pouze jedna součástka. Zároveň pro správné rozpoznávání se musely narovnat nožičky nebo by měly být zastřižené a narovnané.

8.3 Použité typy rezistorů

V této diplomové práci se použily tři druhy rezistorů. Dva typy byly čtyř-pruhové a jeden pěti-pruhový. Na (Obr. 8.1) jsou vyobrazeny typy rezistorů, na kterých se algoritmy testovaly.

Obr. 8.1: Ukázka použitých typů rezistorů

V tabulce 8.1 vidíme vlastnosti jednotlivých typů rezistorů. Je patrné, že třetí typ byl nejmenší a pruhy byly blíže sobě, což může způsobit problém při čtení barevného značení. Čtyř-pruhové rezistory byly lépe čitelné, protože jejich povrch nebyl lesklý a světlo se od nich odráželo v menší míře. Třetí typ rezistorů se nepodařilo sehnat se stříbrnými a zlatými pruhy. Proto jsou výsledky v tabulce 8.4 u těchto pruhů uvedeny jako 0.

Tabulka 8.1: Vlastnosti jednotlivých typů rezistorů

	1. typ	2. typ	3. typ
Vel. těla (š x v) [cm]	1 x 0,3	1,2 x 0,4	0,5 x 0,2
Matné pozadí	Ano	Ano	Ne
Barva pozadí	Tmavě modrá	Šedá	Světle zelená
Počet pruhů	4	4	5

Pochopitelně se lépe četly a rozpoznávaly součástky, které měly tělo větší, byly matné a barva pozadí byla tmavší.

8.4 Nalezení ROI

Nalezení obrysů součástky je spolehlivé a přibližuje se k 90 %. Neúspěch může způsobit jen rozostření objektu nebo špatné osvětlení, které by mohlo vrhat stín. V takovém případě pak algoritmus jako obrys považuje i stín. V případě rozostření rezistoru algoritmus nalezne několik nesouvislých kontur. V mém algoritmu se bere největší souvislá kontura. Tím se vyhneme problémům s obrysy, které by mohly sehrát určitou roli v obraze např. zrnka prachu. Detekce oblasti zájmu při dodržení určitých podmínek, které jsou uvedeny již v předešlé kapitole, tak prvotní doba, kdy algoritmus obraz vyřízne, je okolo 3 s. Následně když uživatel drží mobilní přístroj stabilně, tak průměrná doba detekce se pohybuje mezi 1-2 s. V této době se zároveň počítá i se zpracováním obrazu jako Houghova transformace, vyhledání těla rezistoru atd.

8.5 Správná detekce těla rezistoru

Úspěšnost detekce těla rezistoru se pohybuje kolem 70 %, když součástka leží horizontálně. Algoritmus pro vyhledávání těla se testoval na všech třech typech rezistorů a nejlépe detekoval čtyř-proužkový rezistor s modrým pozadím. Zároveň se ukázalo, že algoritmus špatně vyhodnocuje tělo rezistoru, když součástka má zahnuté nožičky. Algoritmus si správně myslí, že na x ose je nejvyšší hustota pixelů v okolí nožiček. Pokud se ale nožičky rezistoru narovnají, tak algoritmus začne správně rozpoznávat pozici. Když je rezistor nakloněný jinak než horizontálně, tak pomocí přímky z Houhovy transformace určíme úhel, který nám přetočí součástku tak, aby ležela horizontálně.

8.6 Světelnost a vyhodnocení barev

Algoritmy se zkoušely při určitých intenzitách osvětlení. V tabulce 8.2, 8.3 a 8.4 můžeme pozorovat výsledky správného vyhodnocování jednotlivých barev. Je zřejmé, že algoritmy lépe rozpoznávaly konkrétní barvy při vyšší světelnosti. Té se dosáhlo právě přisvětlovací diodou.

Testy probíhaly za třech různých intenzitách osvětlení. První test probíhal při vypnuté přisvětlovací diodě. U druhého testu byla přisvětlovací dioda zapnutá a poslední zkouška proběhla v kombinaci, kdy byla zapnutá přisvětlovací dioda a lampička, která slouží pro osvětlení rybiček, čili její vlnová délka je podobná dennímu světlu.

Tabulka 8.2: Výsledky správné detekce barev vůči celkovému počtu testování – první typ rezistoru

Barva/Světelnost	215 lx	704 lx	1125 lx
Černý	4/10	6/10	5/10
Hnědý	3/10	3/10	6/10
Červený	2/10	4/10	6/10
Oranžový	5/10	5/10	8/10
Žlutý	2/10	4/10	5/10
Zelený	1/10	4/10	4/10
Modrý	2/10	6/10	4/10
Fialový	3/10	3/10	5/10
Šedý	2/10	4/10	5/10
Bílý	3/10	5/10	6/10
Zlatý	1/10	2/10	1/10
Stříbrný	2/10	2/10	3/10

Tabulka 8.3: Výsledky správné detekce barev vůči celkovému počtu testování – druhý typ rezistoru

Barva/Světelnost	215 lx	704 lx	1125 lx
Černý	4/10	6/10	7/10
Hnědý	4/10	4/10	5/10
Červený	2/10	3/10	3/10
Oranžový	2/10	4/10	5/10
Žlutý	3/10	5/10	6/10
Zelený	2/10	5/10	5/10
Modrý	1/10	7/10	6/10
Fialový	2/10	5/10	5/10
Šedý	2/10	4/10	6/10
Bílý	3/10	4/10	5/10
Zlatý	1/10	3/10	2/10
Stříbrný	1/10	2/10	3/10

Tabulka 8.4: Výsledky správné detekce barev vůči celkovému počtu testování – třetí typ rezistoru

Barva/Světelnost	215 lx	704 lx	1125 lx
Černý	4/10	5/10	5/10
Hnědý	3/10	3/10	4/10
Červený	2/10	2/10	3/10
Oranžový	2/10	4/10	3/10
Žlutý	2/10	3/10	3/10
Zelený	1/10	2/10	2/10
Modrý	2/10	4/10	3/10
Fialový	3/10	5/10	6/10
Šedý	2/10	2/10	2/10
Bílý	2/10	3/10	4/10
Zlatý	0/10	0/10	0/10
Stříbrný	0/10	0/10	0/10

9 Možnosti dalšího vývoje

V této diplomové práci se podařilo najít obrys součástky v obraze v poměrně krátkém čase a vysoké úspěšnosti. Algoritmus zároveň počítá s drobnými nečistotami, protože je brán největší obrys z pole. Následně se kontroluje velikost kontury. Tím zabráníme, že se nám budou detekovat obrysy jen části těla rezistoru. Na základě tohoto obrysu lze definovat oblast zájmu, kde se elektronická součástka nachází celá (včetně nožiček). Pomocí Houghovy transformace je možné detekovat úhel naklonění rezistoru. Následně se podaří získat tělo elektronické součástky a z ní vyčítat barvy.

Můžeme zde nalézt testy, kdy se vyhodnocovala správnost detekování jednotlivých barev za různých podmínek, jako byla intenzita osvětlení, odlišné typy rezistorů či pozadí.

Barevné značení není výrobci nijak sjednocené či standardizované. Z toho lze vyvodit, že výrobci mohou značit své součástky i různými odstíny barev. Vzhledem k tomu, že se v této práci používá statická tabulka referenčních barev, tak není možné program používat na rezistorech od různých výrobců.

Typy rezistorů nesou velkou roli při testování, protože vhodně zvolený rezistor může vykazovat daleko lepší výsledky. Jsou to takové rezistory, které mají velikost barevných pruhů široké a pozadí rezistoru ideálně matné. Barevný pruh se vnímá jako barevný gradient, který se může díky okolním podmínkám špatně vyhodnotit, protože gradient bude obsahovat malou četnost skutečné hodnoty.

Problémem se stala i vzdálenost, pokud kolísala, tak algoritmus nevyhodnocoval správně barvy. V případě programu ResCan byl toto hlavní důvod, proč se původní kódy přenášely na desktop, kde pomocí mechanických nástrojů se webkamera umístila do konstantní vzdálenosti vůči rezistoru. V případě, že se mobilní přístroj zachytil do držáku a telefon se nepohyboval, tak výsledky byly věrohodnější.

Další hlavní nevýhodou bylo, že se aplikace testovala v domácích podmínkách a nepodařilo se sehnat lepší osvětlení, protože v článcích a aplikacích se programy testují při vyšší světelnosti. Ukázalo se, že jedna přisvětlovací dioda na mobilním přístroji je nedostatečná. V obraze lze i lidským vnímáním pozorovat, že bílá barva se jeví při nedostatečné intenzitě osvětlení jako šedá z důvodu toho, že na elektronickou součástku zároveň dopadá stín mobilního zařízení.

Pro další vývoj aplikace by bylo dobré najít algoritmus pro rozpoznávání barev natolik inteligentní, že by vhodně rozpoznal značení od různých výrobců. To by možná šlo udělat tak, že by se algoritmus snažil dynamicky přečíst barvy, vyhodnotit je a vložit je správně do referenčních tabulek. Další možností, jak se vyhnout chybám způsobených nejednotností značení od výrobců je, že by se vkládal například XML soubor uživatelem, který by definoval tabulku referenčních hodnot. Zároveň při použití určitých korektur v obraze nebo použití mobilního zařízení, které disponuje více přisvětlovacími diodami, by mohlo dojít k lepšímu vnímání barev v obraze.

10 Závěr

V úvodu práce seznamuji se značením elektrotechnických součástek, konkrétně se jedná o rezistory a kondenzátory. Detailně zde uvádím, k jakým barvám se přiřazují konkrétní hodnoty, protože na této skutečnosti aplikace pracuje.

Druhá kapitola pojednává o principech, jak se vnímá rozdíl mezi dvěma barvami, možnosti převedení do barevného prostoru, který by mohl být lépe vnímán lidským okem nebo jak chápat toleranci k dané barvě.

Následně uvádím aplikace, které pracují na stejném či podobném principu. Popisuji, jak jednotlivé aplikace fungují, a zveřejňuji výsledky jejich testování.

V další kapitole seznamuji s mobilní platformou Android, která se v dnešní době těší velké popularitě. Popisuji dnes používané verze, možnosti vývoje pro tento operační systém a nástroje, které se používají v této diplomové práci.

Na konec teoretické části se zaměřuji na postup při obrazovém zpracování digitálních snímků. Uvádím zde princip a použití barevných modelů, princip a srovnání hranových a rohových detektorů a dalších technik v oblasti zpracování obrazu, jako Houghova transformace či Blob detekce.

V praktické části navazuji na teoretické základy a vysvětluji, jak je aplikace složena a představuji strukturu a grafickou část aplikace. Zároveň detailně popisuji implementaci jednotlivých kroků aplikace včetně ukázkových fotografií a jejich vzájemnou návaznost, která vede až ke konečnému dopočítání ohmické hodnoty elektronické součástky.

Dále je uvedena kapitola, která pojednává o testování a dosažených výsledcích. Tomuto kroku bylo nutné věnovat nejvíce času, protože na základě praktické ukázky se prokázalo, že mnoho teoretických informací funguje v praxi odlišně, a proto se musely segmenty aplikace různě přepisovat a upravovat. Popisuji i problémy, se kterými jsem se během tvorby při vývoji této aplikace setkal.

Poslední kapitola shrnuje dosažené výsledky, kriticky hodnotí jednotlivé části aplikace a otevírá možnost pro další vývoj.

Aplikací, které řeší podobný problém, není mnoho. Většina z nich byla již stažena z nejrůznějších důvodů nebo se přenesly z mobilních zařízeních na stolní počítače. To se stalo hlavním důvodem, proč si vytvořit vlastní aplikaci, která se snaží pomocí vhodných technik dosáhnout dobrého výsledku.

Program oproti svému předchůdci funguje real-time, takže uživateli usnadňuje manipulaci a ušetřuje ho zbytečných kroků. V praxi je ale závislá na provedení elektronické součástky a na jejím značení. Dále je nutné počítat s tím, že některé barvy jsou svými vlnovými délkami tak blízko sebe, že dělají problém i lidskému vnímání. Toto tvrzení lze vidět v referenční tabulce mezi barvou fialovou a šedou.

Další vývoj aplikace by mohl znamenat, že by se úspěšnost vyhodnocování mohla zvýšit.

Použitá literatura

[1] *A guide to Understanding Color Tolerancing*. Michigan USA, 1997. Dostupné z: http://www.xrite.com/documents/literature/en/L10-024_Color_Tolerance_en.pdf

[2] An introduction to Text-To-Speech in Android. Android Developers Blog [online]. 2009 [cit. 2014-03-07]. Dostupné z: http://android-developers.blogspot.cz/2009/09/introduction-to-text-to-speech-in.html

[3] Android: A visual history. The Verge [online]. 2014 [cit. 2014-03-07]. Dostupné z: http://www.theverge.com/2011/12/7/2585779/android-history

[4] ALLEN, Grant. Android 4: průvodce programováním mobilních aplikací. 1. vyd. Brno: Computer Press, 2013, 656 s. ISBN 978-80-251-3782-6.

[5] BEZDĚK, Miloslav. Elektronika: [učebnice]. 1. vyd.. České Budějovice: Kopp, 2004, 286 s. ISBN 80-723-2171-4.

[6] BRÄHLER, Stefan. Analysis of the Android Architecture. Karlsruhe, 6. 10. 2010. Dostupné z: http://pdf-world.net/pdf/3183/Analysis-of-the-Android-Architecture-pdf.php. Diplomová práce. Karlsruher Institut für Technologie. Vedoucí práce Prof. Dr. Frank Bellosa.

[7] BURGER, Wilhelm a Mark BURGE. *Principles of digital image processing: core algorithms*. London: Springer, c2009, xii, 327 p. Undergraduate topics in computer science. ISBN 18-480-0190-8.

[8] BURGER, Wilhelm. *Digital image processing: an algorithmic introduction using Java*. 1st ed. New York: Springer, c2008, xx, 564 s. ISBN 978-1-84628-379-6.

[9] BURNETTE, Ed. Eclipse IDE: pocket guide. 1st ed. Sebastopol, CA: O'Reilly, c2005, ix, 117 p. ISBN 05-961-0065-5.

[10] Camera. Android Developers [online]. 2014 [cit. 2014-03-07]. Dostupné z: http://developer.android.com/guide/topics/media/camera.html

[11] Digital imaging/Image analysis. *LabAutopedia* [online]. 2013 [cit. 2014-04-19]. Dostupné z: http://www.labautopedia.com/mw/index.php?title=Digital_imaging/Image_analysis#Blob_analysis

[12] Delta E (CIE 1994). *Welcome to Bruce Lindbloom's Web Site* [online]. 2011 [cit. 2014-04-27]. Dostupné z: http://www.brucelindbloom.com/index.html?Eqn_DeltaE_CIE94.html

[13] Delta E (CIE 2000). *Welcome to Bruce Lindbloom's Web Site* [online]. 2009 [cit. 2014-04-27]. Dostupné z: http://www.brucelindbloom.com/index.html?Eqn_DeltaE_CIE2000.html

[14] Getting Started with Android Studio [online]. 2014 [cit. 2014-03-07]. Dostupné z: http://developer.android.com/sdk/installing/studio.html

[15] GUNASEKERA, Sheran A. Android apps security. New York, NY: Apress, 2012. ISBN 978-143-0240-631.

[16] HÁJOVSKÝ, Radovan, Radka PUSTKOVÁ a František KUTÁLEK. *Zpracování obrazu v měřící a řídící technice*. Vyd. 1. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2012, 1 DVD-ROM. ISBN 978-80-248-2596-0.

[17] HLAVÁČ, Václav a Miloš SEDLÁČEK. *Zpracování signálu a obrazu*. Praha, 1999. Dostupné z: http://neuron.tuke.sk/pluchta/Pocitacove%20Videnie/Prednasky/NIECO/HLAZSO.PDF

[18] IGNAC LOVREK, Robert J. *Knowledge-based intelligent information and engineering systems. 12th international conference, KES 2008, Zagreb, Croatia, September 3-5, 2008: proceedings*. Berlin: Springer, 2008. ISBN 978-354-0855-637.

[19] JACK, Keith. *Video demystified: a handbook for the digital engineer*. 5th ed. Boston: Newnes, c2007, xix, 920 p. ISBN 07-506-8395-3.

[20] Jmenovité hodnoty odporů a kondenzátorů. *Elektronika: Teoretické zákaldy, praktická zapojení* [online]. 2004 [cit. 2014-04-19]. Dostupné z: http://elnika.sweb.cz/mer_sou/jmen_hodn_odpor.htm

[21] KONG, Hui, Hatice Cinar AKAKIN a Sanjay E. SARMA. A Generalized Laplacian of Gaussian Filter for Blob Detection and Its Applications. *IEEE Transactions on Cybernetics*. 2013, vol. 43, issue 6, s. 1719-1733. DOI: 10.1109/TSMCB.2012.2228639. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6408211

[22] MURPHY, Mark L. Android 2: průvodce programováním mobilních aplikací. Vyd. 1. Překlad Jakub Mužík. Brno: Computer Press, 2011, 375 s. ISBN 978-802-5131-947.

[23] OLIVEIRA, V. A. a A. CONCI. Algorithm for skin detection. *Skin Detection using HSV color space*. 2010, č. 12. Dostupné z: http://www.matmidia.mat.puc-rio.br/sibgrapi/media/posters/59928.pdf

[24] *OpenCV* [online]. 2014 [cit. 2014-04-26]. Dostupné z: http://opencv.org/

[25] ResCan — Automated Resistor Identification!. *Hackaday — Fresh hacks every day* [online]. 2014 [cit. 2014-04-26]. Dostupné z: http://hackaday.com/2014/02/16/rescan-automated-resistor-identification/

[26] Resistor Photo ID. *Nothing Labs (Rich Olson)* [online]. 2011 [cit. 2014-04-26]. Dostupné z: http://nothinglabs.blogspot.cz/2011/11/resistor-photo-id.html

[27] RGB to HSV color conversion. *Online Reference & Tools - RapidTables.com* [online]. 2014 [cit. 2014-04-19]. Dostupné z: http://www.rapidtables.com/convert/color/rgb-to-hsv.htm

[28] SHARMA, Gaurav. *Digital color imaging handbook*. Boca Raton, FL: CRC Press, c2003, 797 p. ISBN 08-493-0900-X.

Seznam příloh

Příloha A: Uživatelský manuál

Instalace

Pro správný běh aplikace je nutné si do mobilního zařízení dopředu nainstalovat z Google Play aplikaci nazývanou OpenCV Manager. Tento program zajistí správný běh aplikace, protože jádro aplikace je postavené na zpracování obrazu využívající stavební prvky právě z knihovny OpenCV.

Fotoaparát

Chcete-li správně získat hodnotu rezistoru, tak se doporučuje mít osvětlení co nejvyšší. Zároveň je nutné, aby se mobilní zařízení vyskytovalo v dostatečné vzdálenosti, protože tak je možné zajistit, aby se rezistor dobře zaostřil. V případě, že se nedaří zaostřit, tak je nutné mobilní zařízení namířit na jiný objekt, počkat, až se objektiv zaostří a zkusit znovu namířit na elektronickou součástku. Výsledek se ukáže a vyhlásí, až se barevné značení detekované mobilním zařízením ustálí.

Příloha B: Adresářová struktura přiloženého DVD

/Zdrojové kódy	Projekt pro vývojové prostředí Eclipse